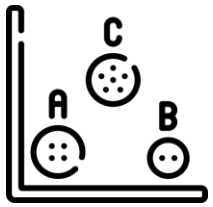
Fake News: Classification & Generative Model

Alok Malla, Carlos Reichenbach de Sousa, Chen Cao, Tangzhi Pang, Mohammadhadi Shahhosseini

Introduction

Project Goals:



Task 1: Clustering

Find the topics in our data

Task 2: Classification

Be able to differentiate between fake and real news

Task 3: Text generation

Create a system capable of generating convincing fake news articles from a given prompt

The Data : **WELFake Dataset**

BIG

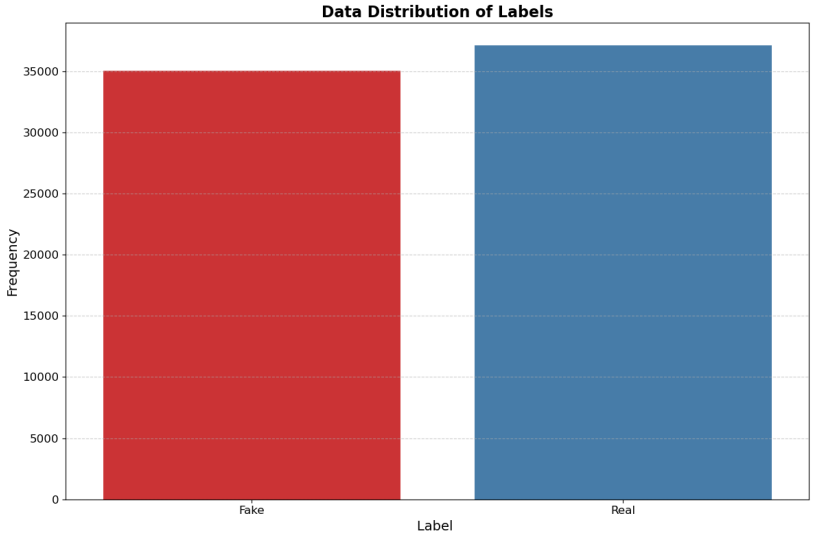
N = 72’134
columns: (index, title, text, label)

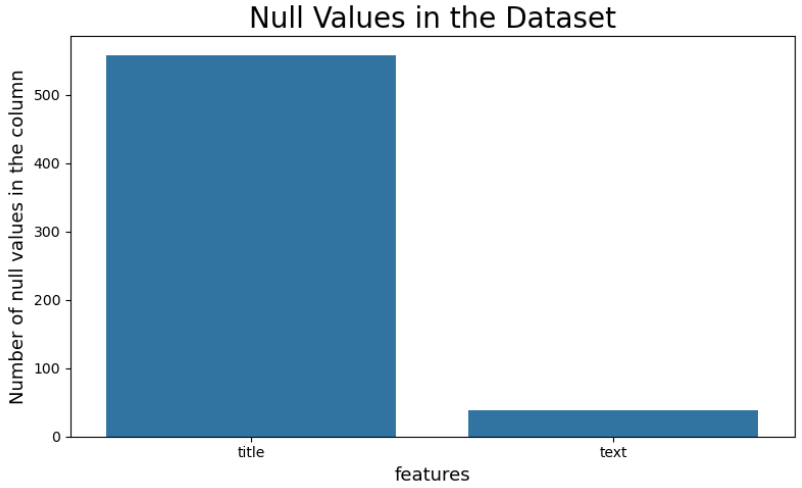
BALANCED

(51.4% fake news vs 48.6% real news)

IMPERFECT

597 entries with a NaN value (0.83%)
9415 duplicate text entries (13.05%)





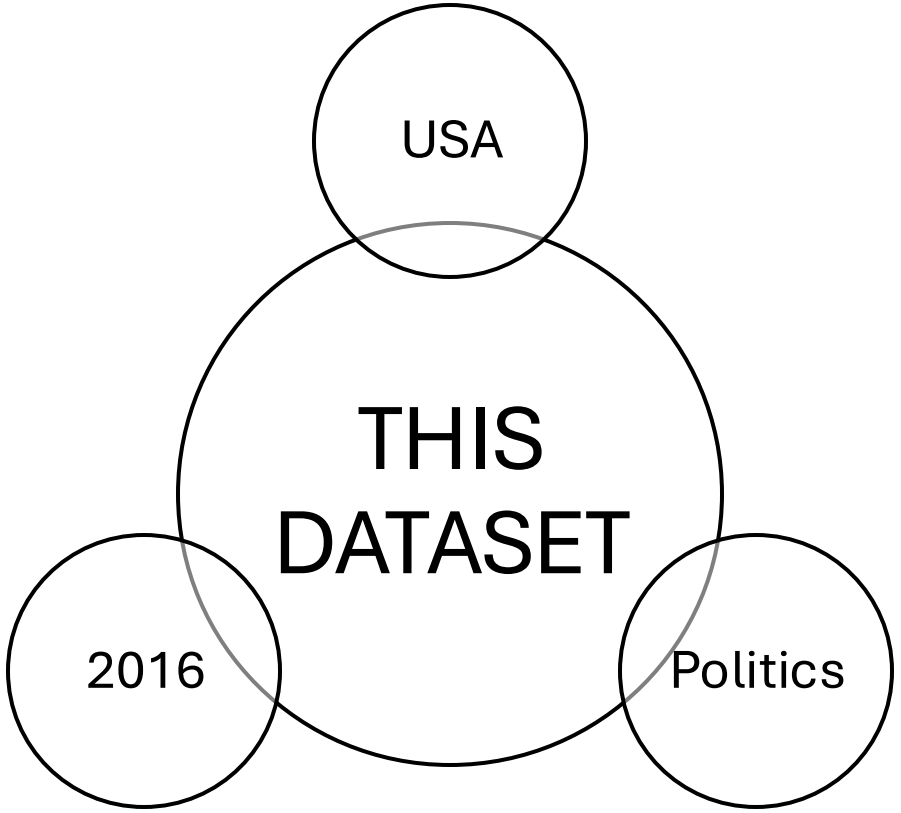
The Data : **Word analysis**

Median text length: 3268

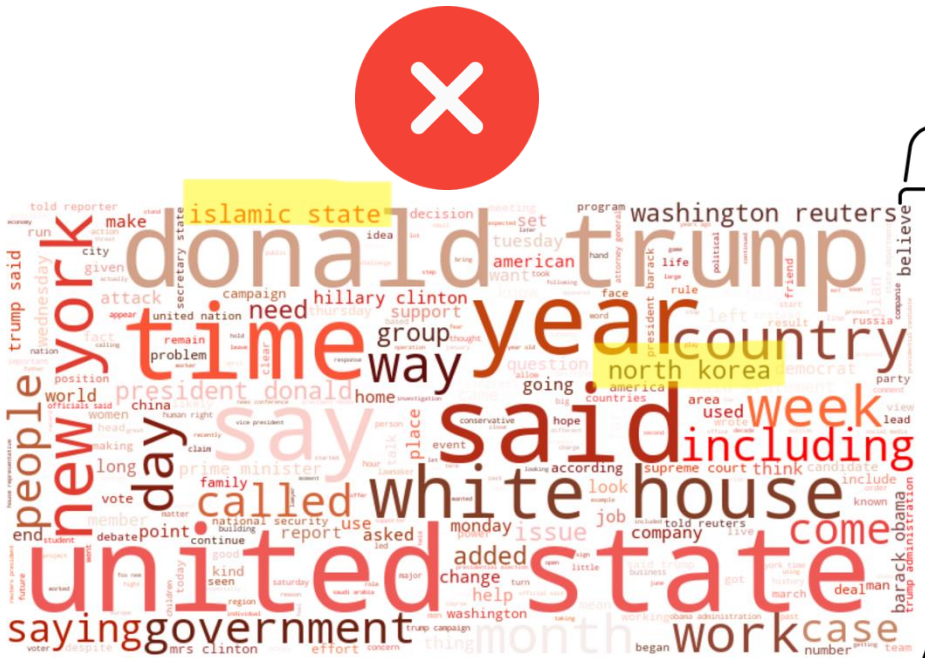
Unique words: 124566

Most frequent words:

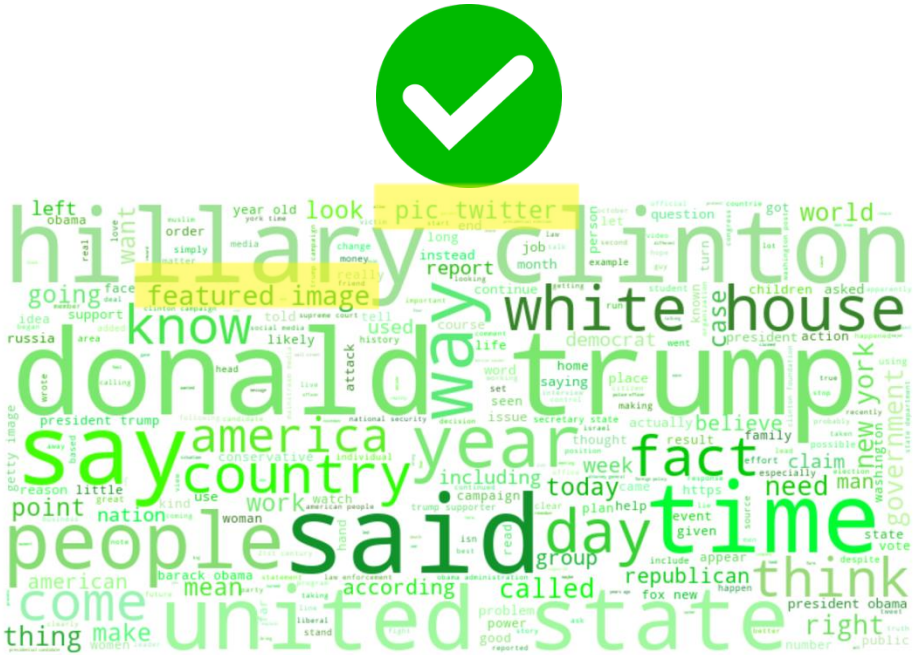
- Donald Trump
- United States
- White house
- Hillary Clinton
- ...



The Data : **Word analysis**

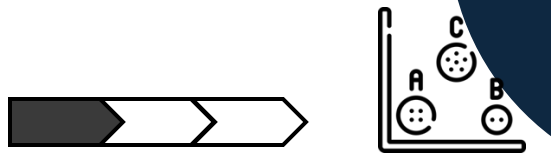


. Worldcloud of full fake dataset

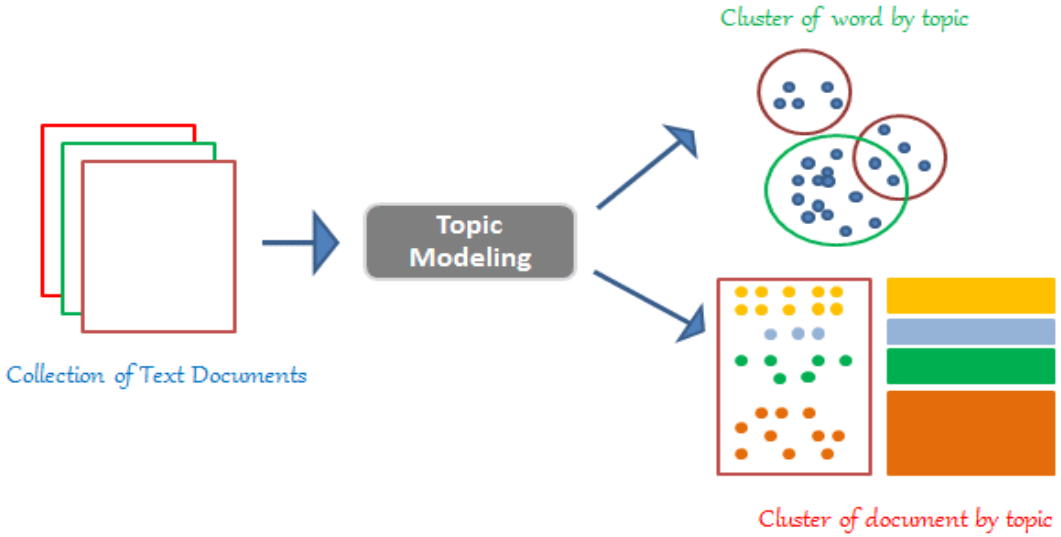


Worldcloud of full real dataset

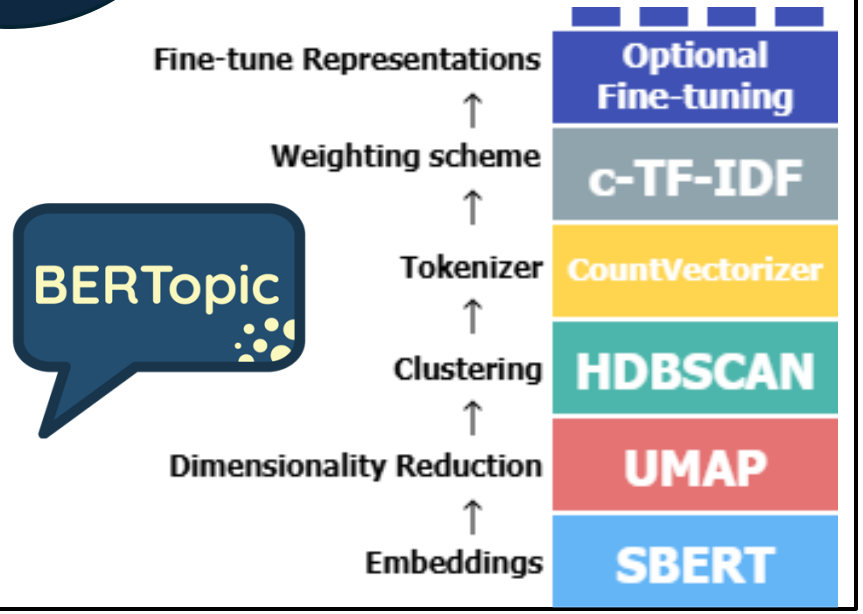
Topic Modeling



Latent Dirichlet Allocation (LDA)



BERTopic



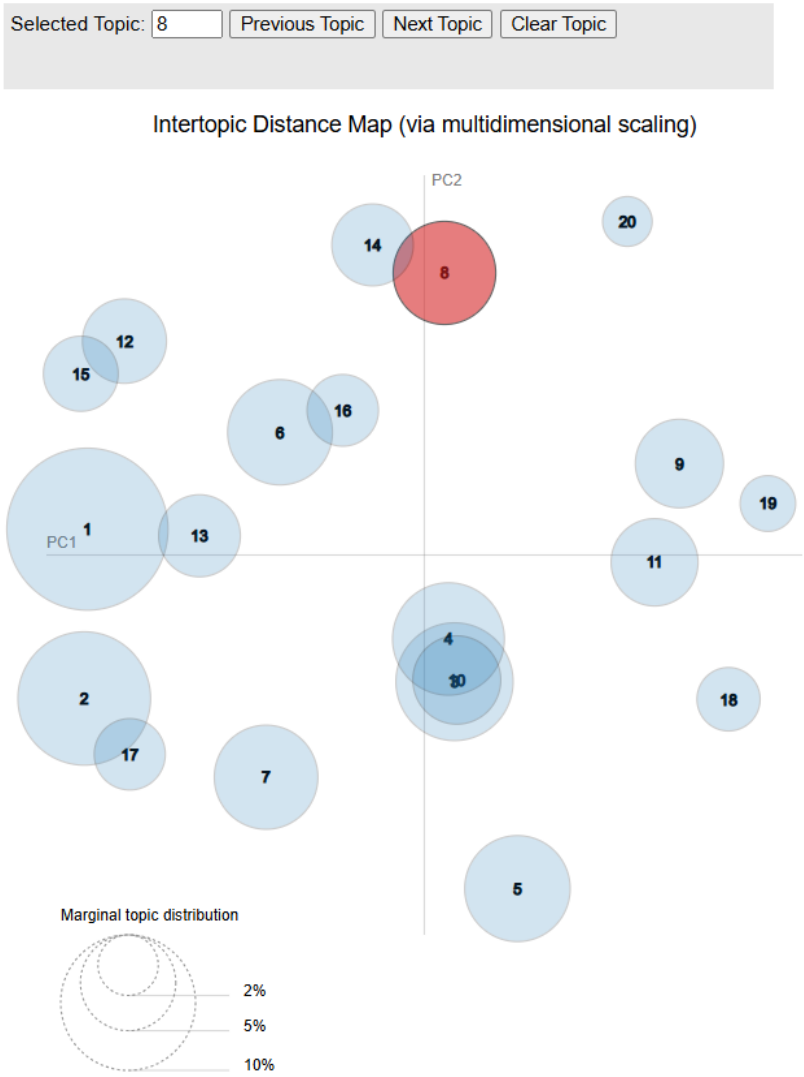
Topic Modeling with LDA

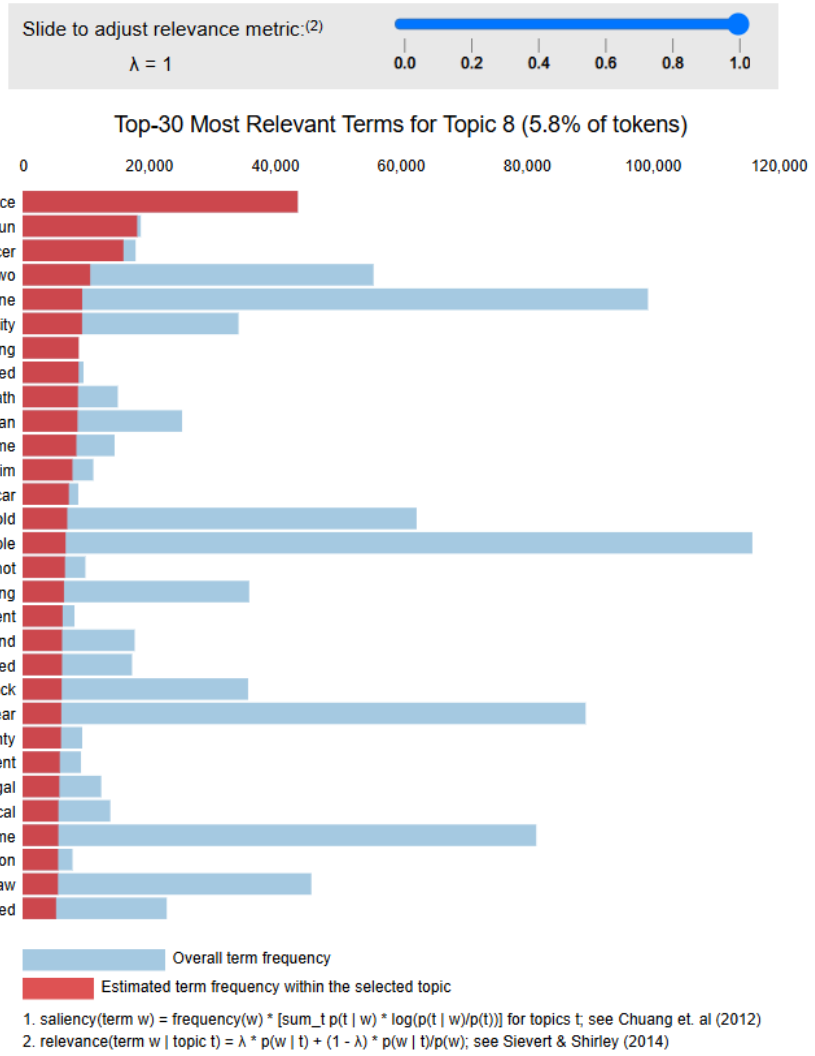
Bubble size:
topic frequency in the dataset

Bubble distance:
similarity between topics

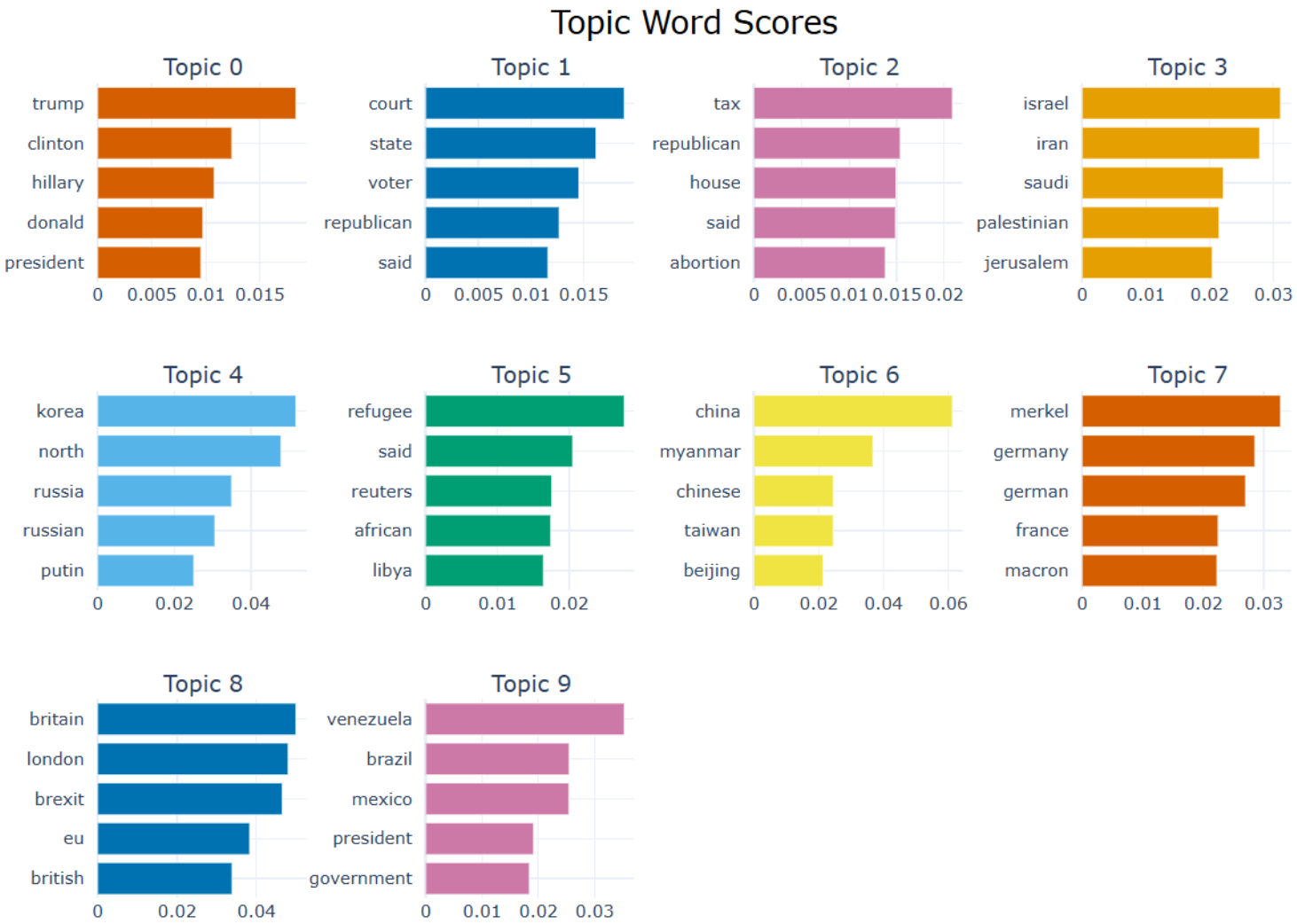
Bar chart:
top keywords for selected topic (e.g., Topic 8)

Example:
Topic 8 → *police, gun, officer*
→ crime-related theme





Topic Modeling with BERTopic



We got about 750 topics and reduced them to 50

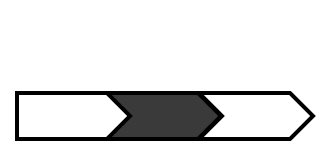
Bar height:
word relevance within topic

Captures **semantics**, not just word counts

More fine-grained and **diverse topics** than LDA

Great for **interpreting context** in real vs. fake news

BERT Classifier



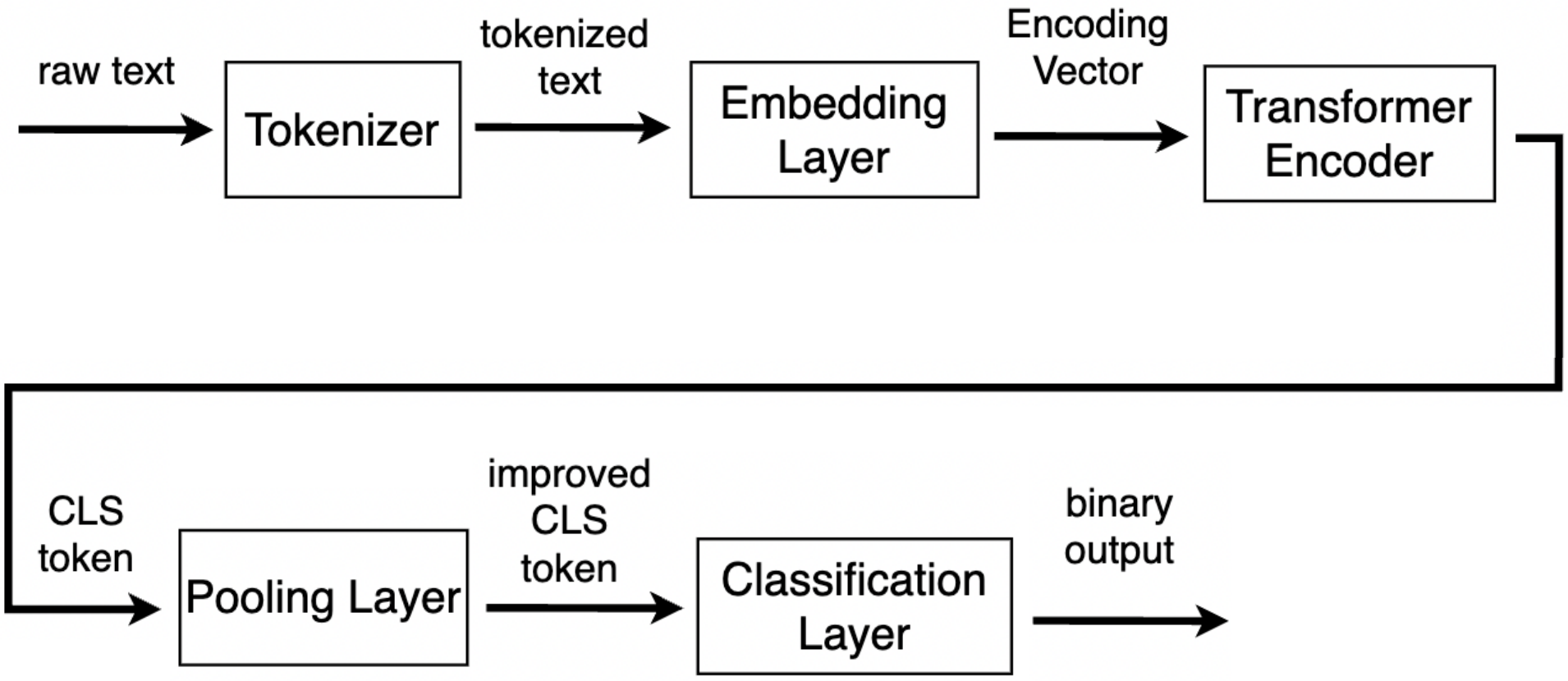
Bidirectional **E**ncoder **R**epresentations from **T**ransformers

- Pre-trained language model
- developed by Google
- Trained on Wikipedia & BookCorpus

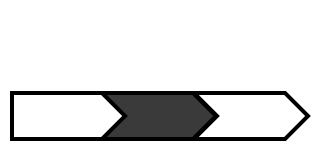
For the Project: BERT Classifier

- BERT Model + classifier layer
- One linear layer on top
- 786 Input nodes, 2 output nodes

BERT Classifier: Structure



BERT Classifier: Training



Data Cleaning

Remove NaNs
Split data at median
48‘302 Entries, 55/45 split

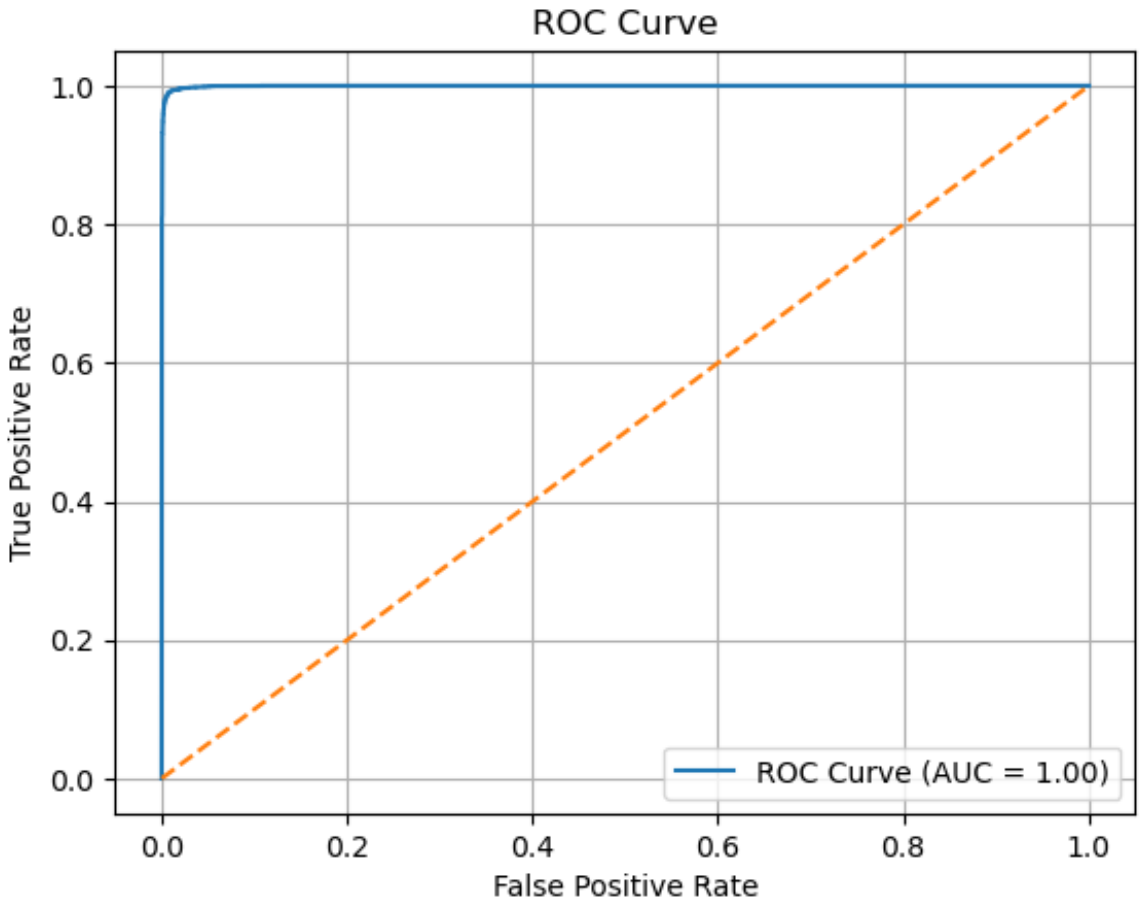
Data Split

80/20 Split
Test set: Different data

Training Time

20 Epochs planned,
5 Epochs done
Around 12 Hours
MacBook Pro (M3)

BERT Classifier: Results on validation Set

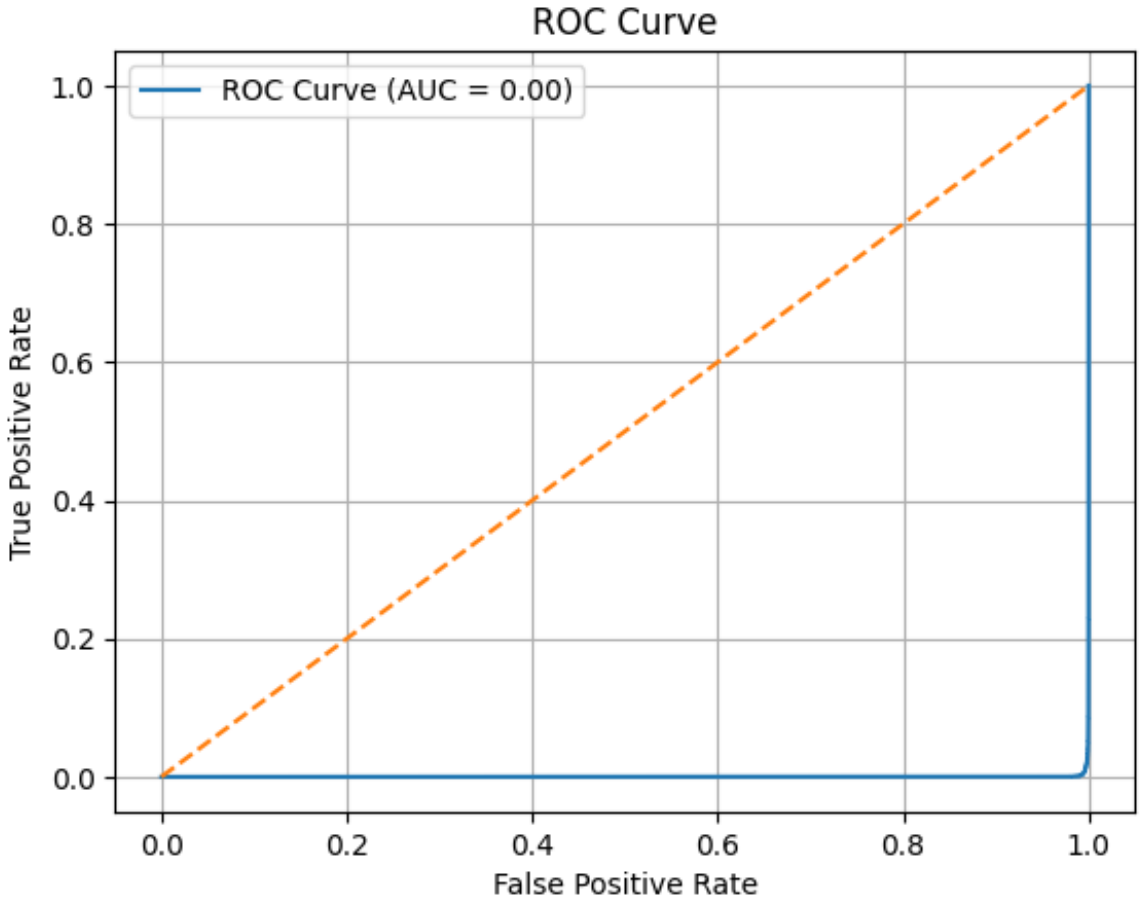


Accuracy on validation Set:
0.9909

Almost perfect ROC curve
Raises suspicions

Possible overfitting

BERT Classifier: Test Set



Test Set:
Different Dataset

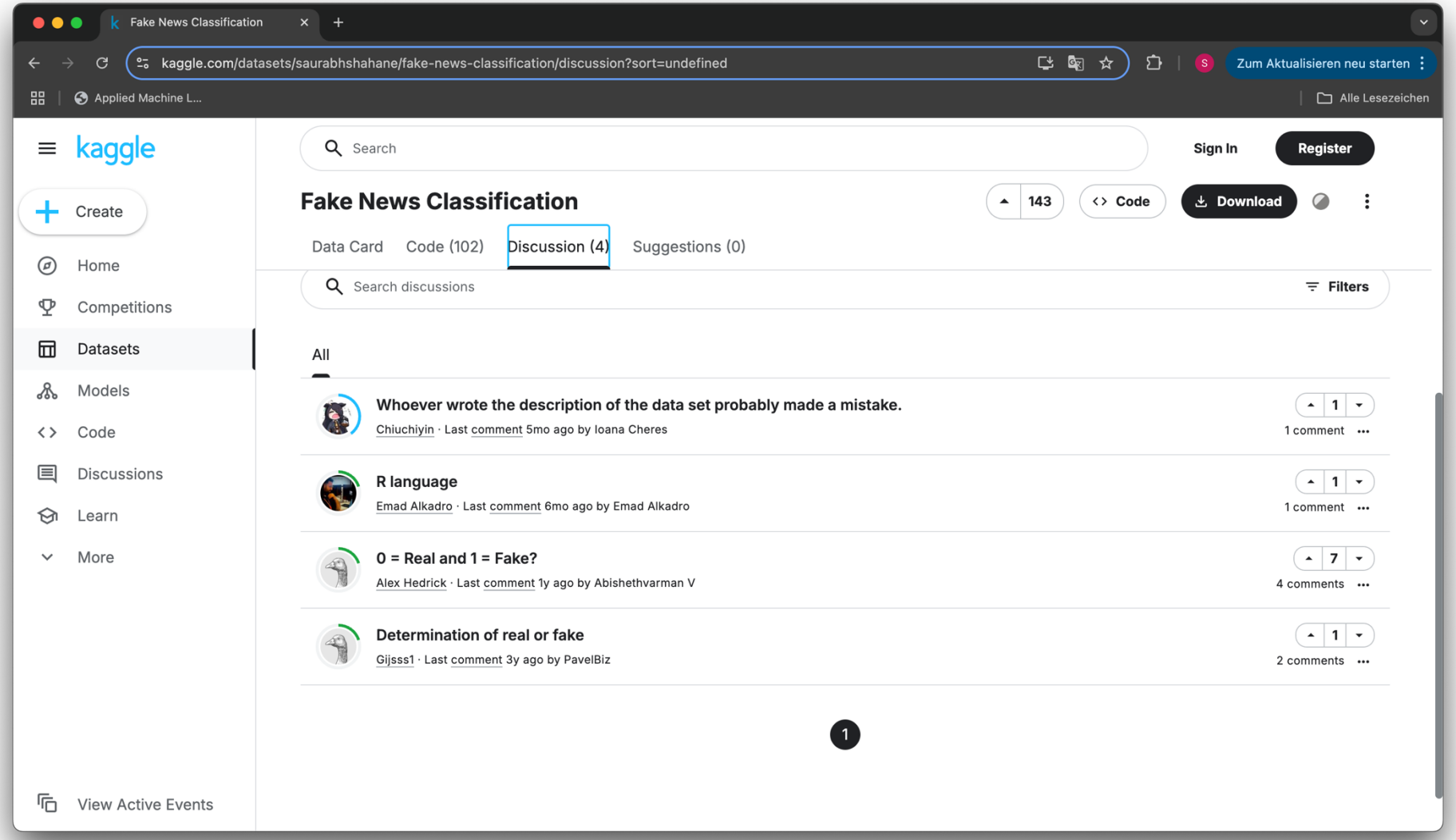
Accuracy on test Set:
0.0104

Terrible Performance

Possible Explanations:
Massive Overfitting

Labels on the Dataset are flipped

BERT Classifier: Discussion about Dataset



BERT Classifier: Conclusions

Performance

Amazing on validation set

Terrible on test set

Future Work

Investigation of dataset

Better data preprocessing

Possible Explanations

Massive overfitting

Wrong labels

Key Learning Points

BERT is a powerful tool for NLP

Check datasets thoroughly

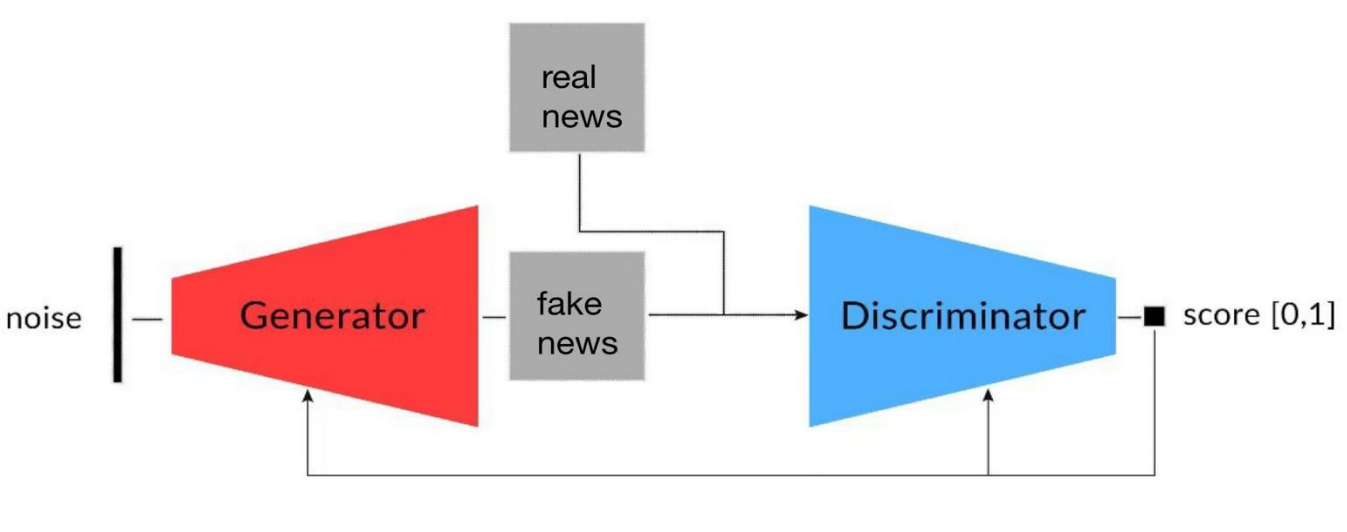
Validation score can be misleading

Fake News Generator

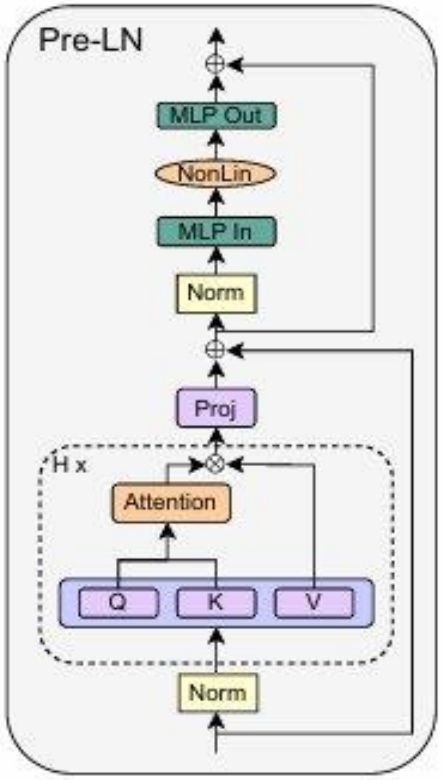
To stress-test the fake/real news classifier to further enhance it.

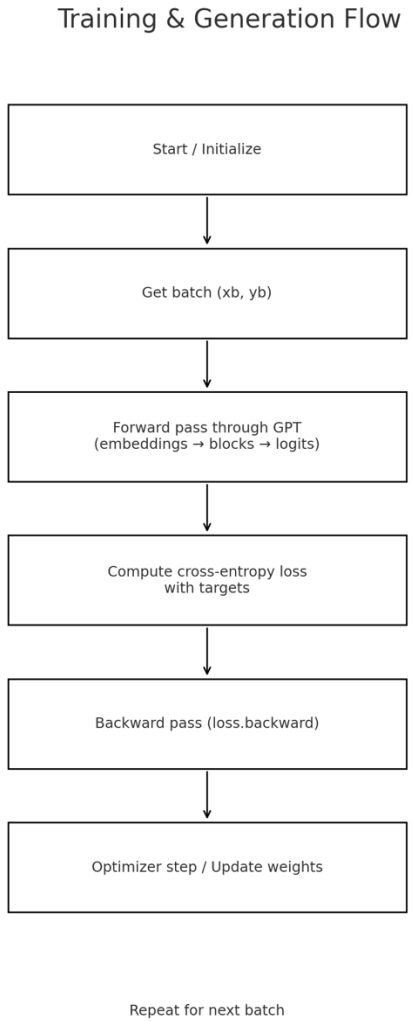
Goal: Generate high-quality, hard-to-distinguish fake news for the classifier.

Generative Adversarial Networks



Training & Generation Flow

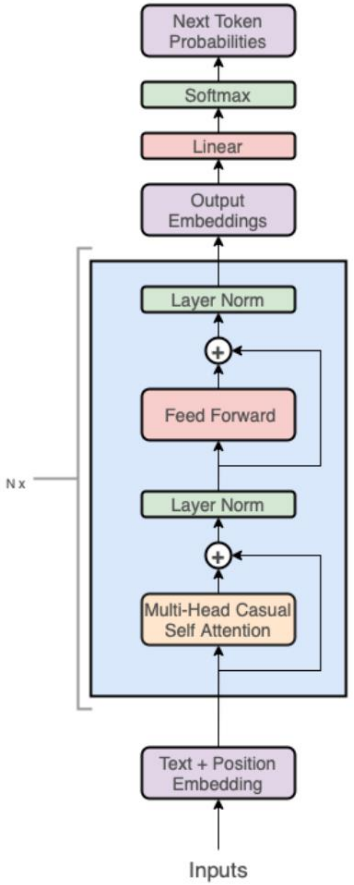


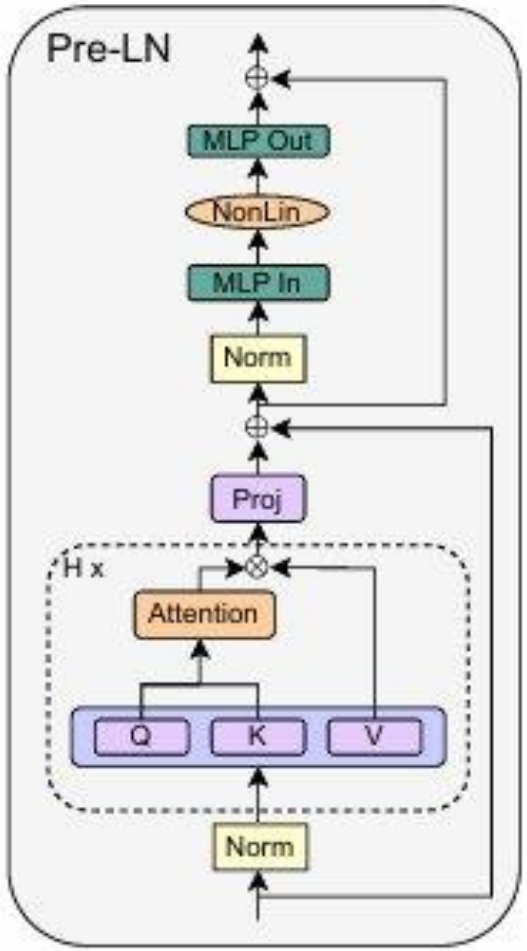


Generation mode:
• Crop context ≤ block_size
• Forward pass
• Temperature & Top-k
• Sample next token
• Append & loop

GPTv1 Architecture

BATCH_SIZE = 64
BLOCK_SIZE = 256
MAX_ITERS = 5000
EVAL_INTERVAL = 250
LEARNING_RATE = 0.0003
N_EMBD = 384
N_HEAD = 6
N_LAYER = 6
DROPOUT = 0.2
BETA1 = 0.9
BETA2 = 0.95
EARLY_STOPPING_PATIENCE = 5
EVAL_ITERS_FOR_LOSS = 100



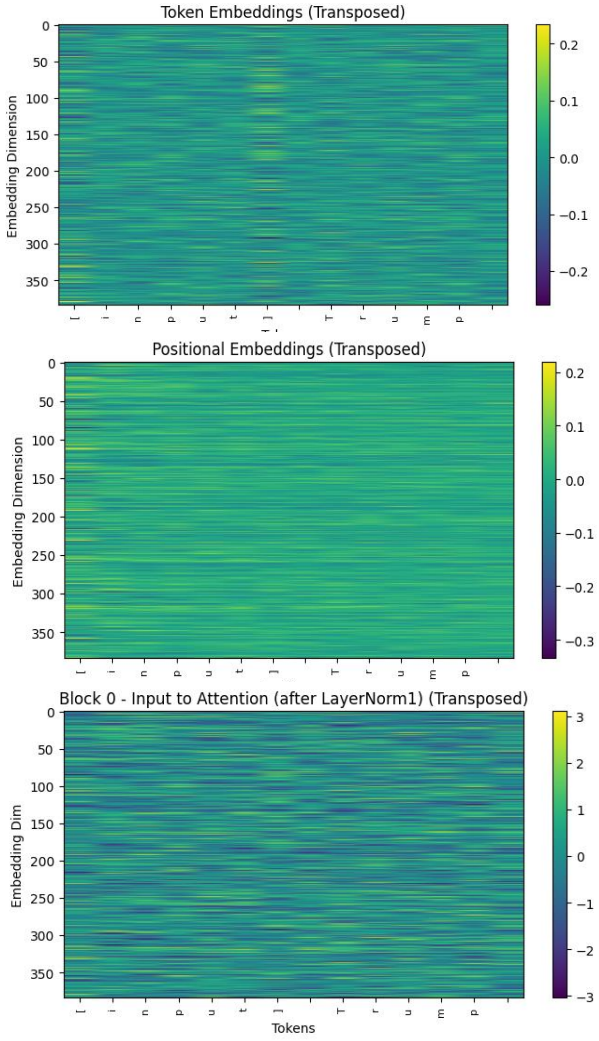


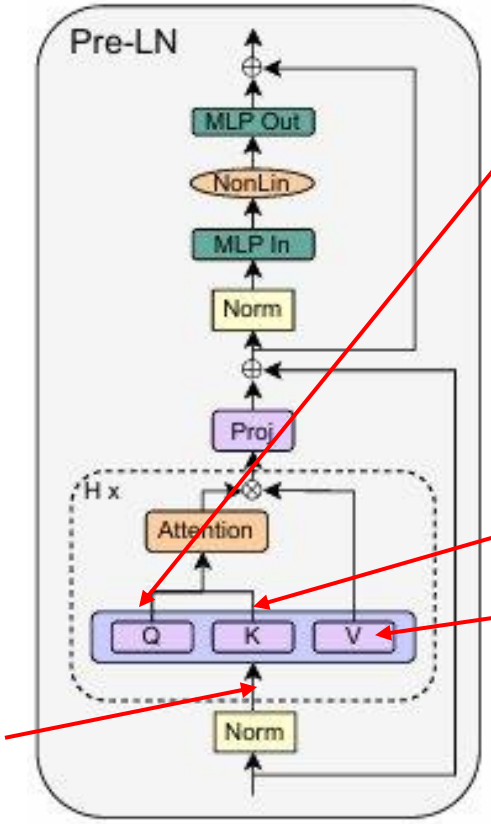
Encoding character as tokens
Vocab_size: 275. Params: 10M

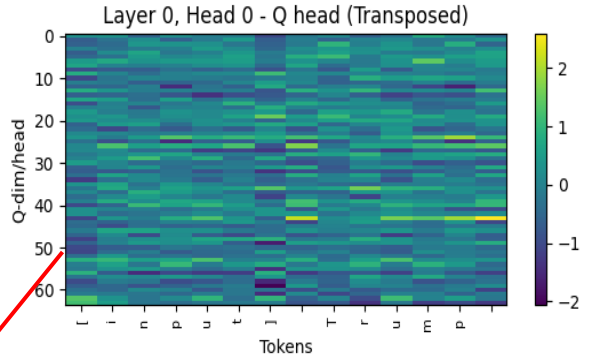
Training time：50 min

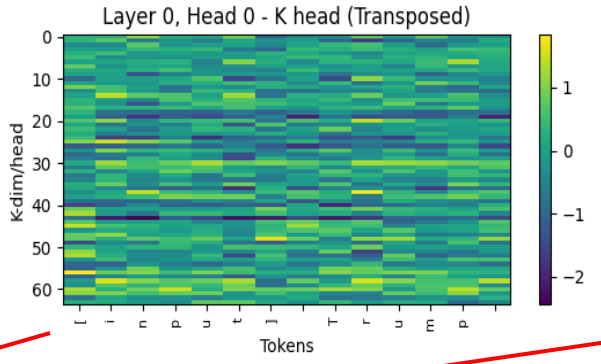
Device: RTX 4060Ti 8+8G Memory

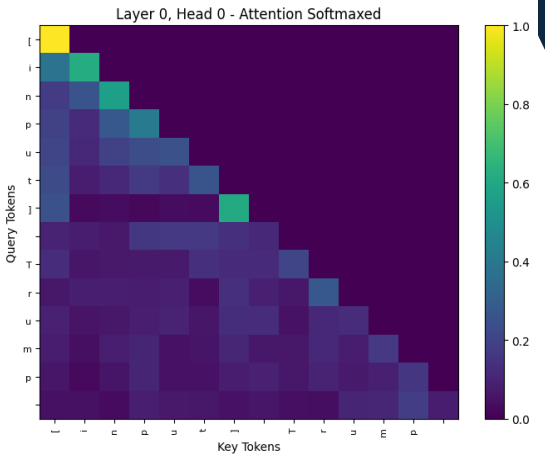
Latent Space Visualization

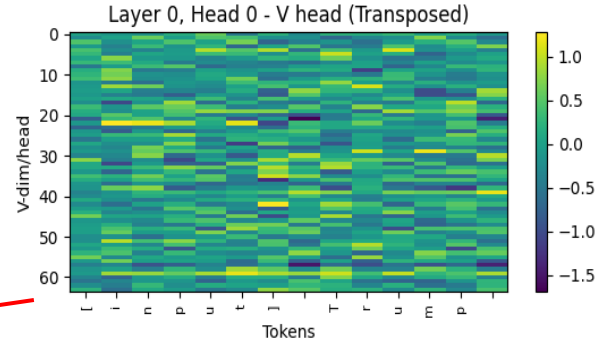


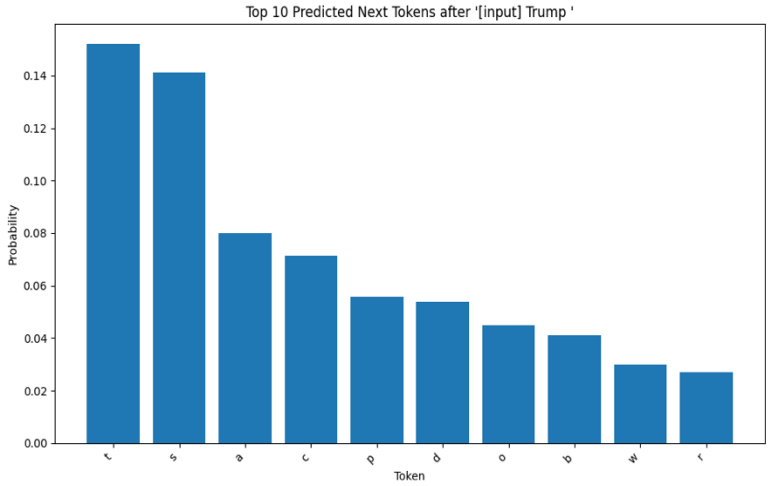








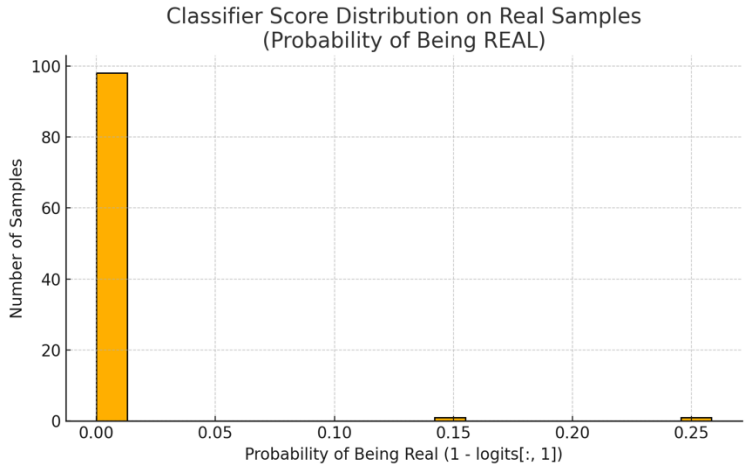




GPT model-v1 Performance Evaluation

- Human Evaluation(Semantic incoherence, Fabricated content, Logical collapse, Inclusion of fictional entities)

- Ability to Evade Classifier Models(Generate 100 fake news, and then scoring by the classifier)



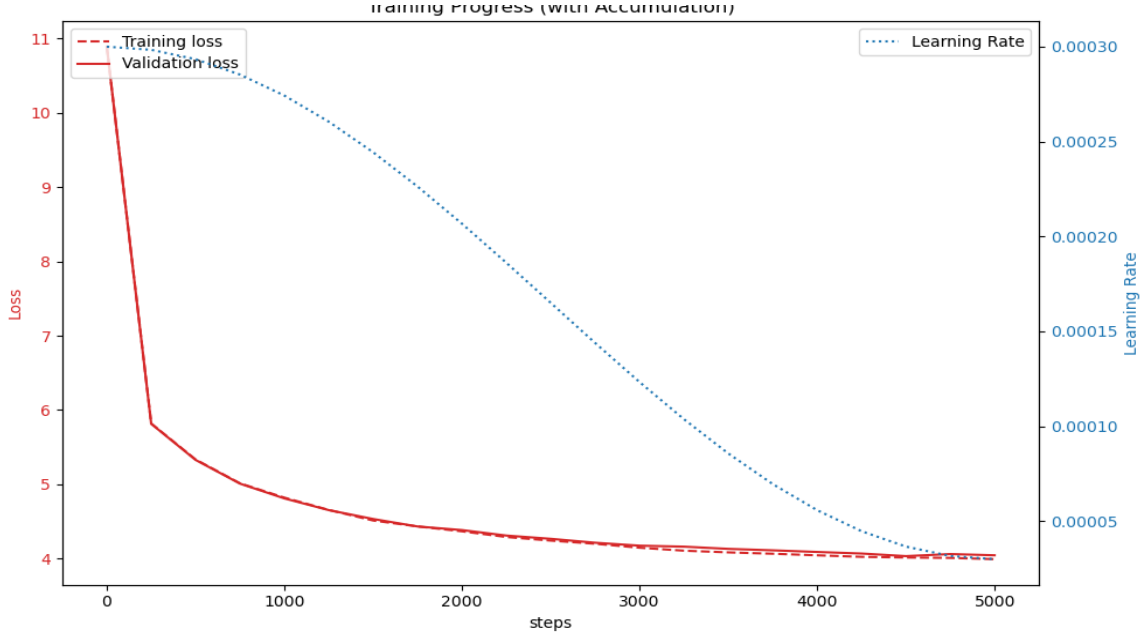
Starting prompt: '[input] Trump'
Generated Text: '[input] Trump says U.S. to be a criticism of Manafort && WASHINGTON (Reuters) - Breitbart News Daily Ericht is also reported on Friday to a political and other citizens to the country s behavior, calling the U.S. President Donald Trump to receive back its strategy to accept a very case for the president by the U.S. Ambassador Ri Embassy to require comment. For the election in which Sanchuary on Sunday, a Sept. 12 state from 1896, in the meeting with Trump's spokesman for the vote on the Supreme Court was revised for an adviser to prosecutors, fired concerns about safety for the nation's embassy in a statement. The Trump administration has been repeated for a Supreme Court in the Supreme Court of Hurricane Intelligence Agency to President Barack Obama approved suspects of the Senate nomination on Friday, which had sought a gun and starting on a U.S. President-elect Donald Trump administration in Afghanistan in 2013. He has said the Iran-backed California and the Muslim decision in 2015 as he would be dealing through the King country in competitors, and more than 100 million ends of dominating from policies. The activists protesters have been suspected since the Sinai news agency against a monthly of north of a charges, marijuana reported. The town of Afghanistan African managers are since 2018 and with a plan to discuss the song alleged budget on the content as the UNATE has become on the conflict, which also called for the report during the revolution of the Syrian civil war.'

Optimized GPT Model-v2

BATCH_SIZE = 32
ACCUM_STEPS = *2*
BLOCK_SIZE = 512
MAX_ITERS = 5000
EVAL_INTERVAL = 250
initial LEARNING_RATE = 3e-4
N_EMBD = 384
N_HEAD = 8
N_LAYER = 8
DROPOUT = 0.1

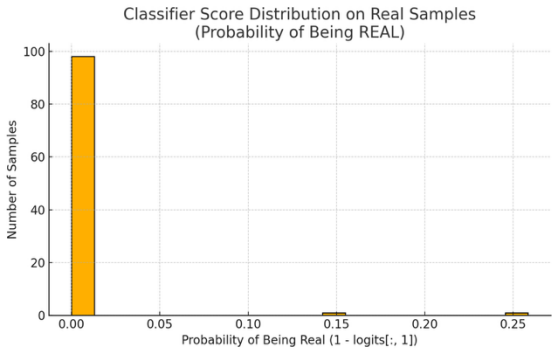
BPE(sub-words as tokens)
Vocab size: 50,257. Params:33M
Training Device RTX 4060ti
Training time ： 11hours

- Training on the very limited GPU
- Automatic Mixed Precision
- Gradient Accumulation

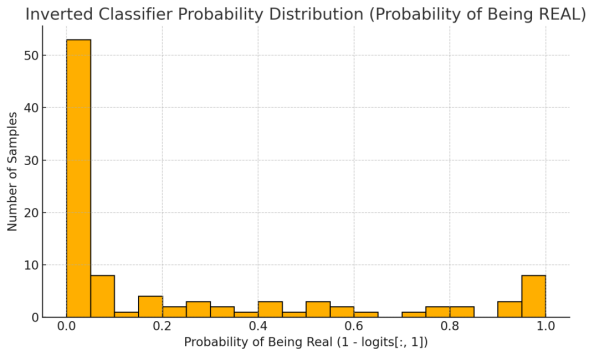


GPT model-v2 Performance Evaluation

- Human Evaluation(Relatively more structured, superficially plausible, still largely fabricated)
- Have ability to evade classifier models (Generate 100 fake news, and then scoring by the classifier)



GPTv1



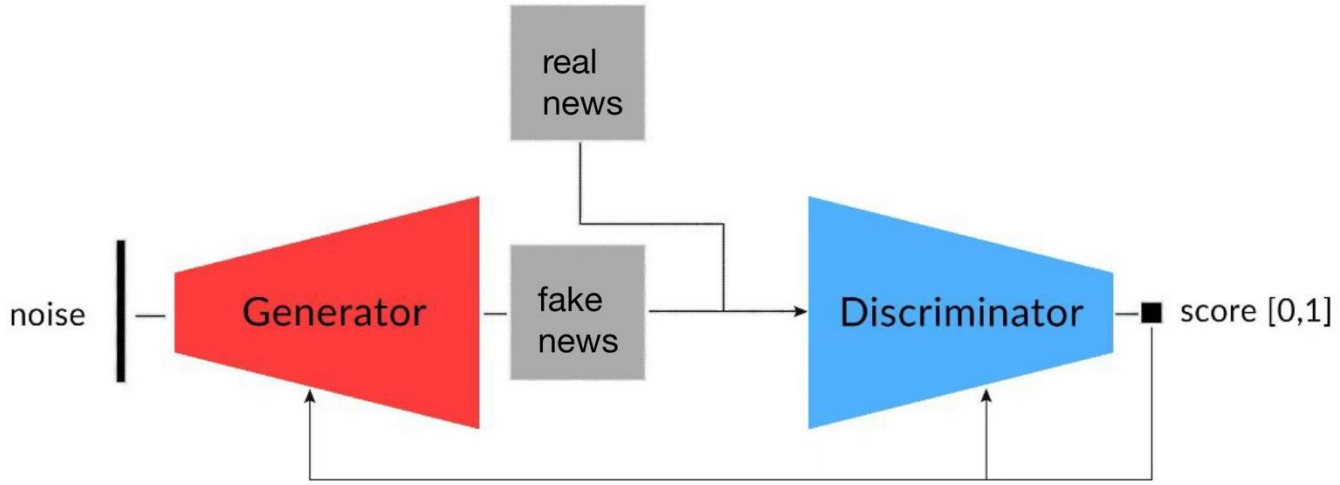
GPTv2

The New Press: Trump: Russia says Putin is about to be the US president ||| Text: November 8th, 2016 Comments Trump is the US president , the “Russia’s Russia. They’ve put all of our sources away.” As we see, the White House says, they reopening their investigation into the Russian hacking of the DNC. Trump did not go wrong. The U.S. election should be a very high-ranking ally in Russia, one that Trump is facing controversy, but so we can see better reports about what happens in Russia. They re investigating and Russia. Putin doesn’t know who he plans to make, even though Russian hackers can easily get into the polls. It’s so late they see, I’m getting the news coverage he says, Trump has shown up his reputation for hacking. It’s in your way that people are actually voting, the fact that Trump made it clear that Russians would have to get out of those states. Trump has promised that ‘for the next few weeks:’ Donald Trump said he knows ‘I think he should ‘no.’ And he doesn’t care, but he wants him to do his job, or even though a lot of other than Trump were doing this. So what’s happening? How are those comments? Trump then claimed that Putin is not going to talk about Russian interference in the election and Trump is going to have any trouble that he was trying to make. Putin also said that Trump is considering the case because he wants to go out to Congress, and when he’s trying to keep him, he will have them out there. He will have a different conversation with Clinton, Putin and Putin to get behind Trump’s election.

Adversarial training between GPT model and Classifier

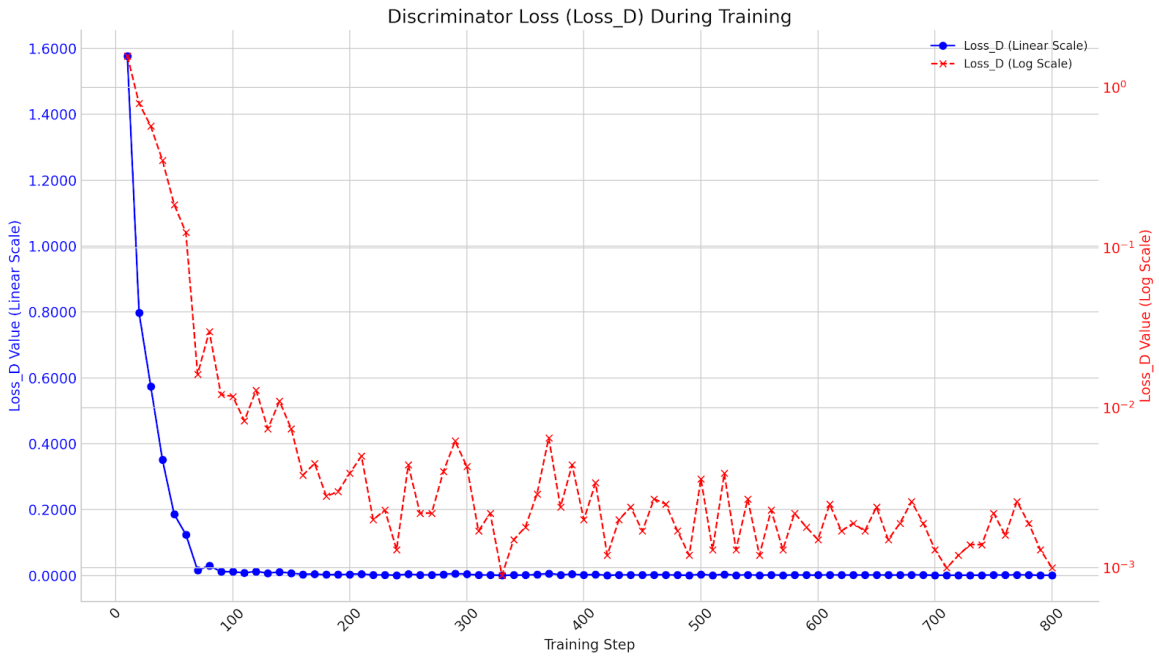
- 100 iterations, 9 hours training
- In training, the loss always large
- Generator started generating low-quality, easy-to-distinguish fake news with garbled characters
- Despite the incoherence, the classifier labels the output from generator as "1" (real)
- It gets worse?

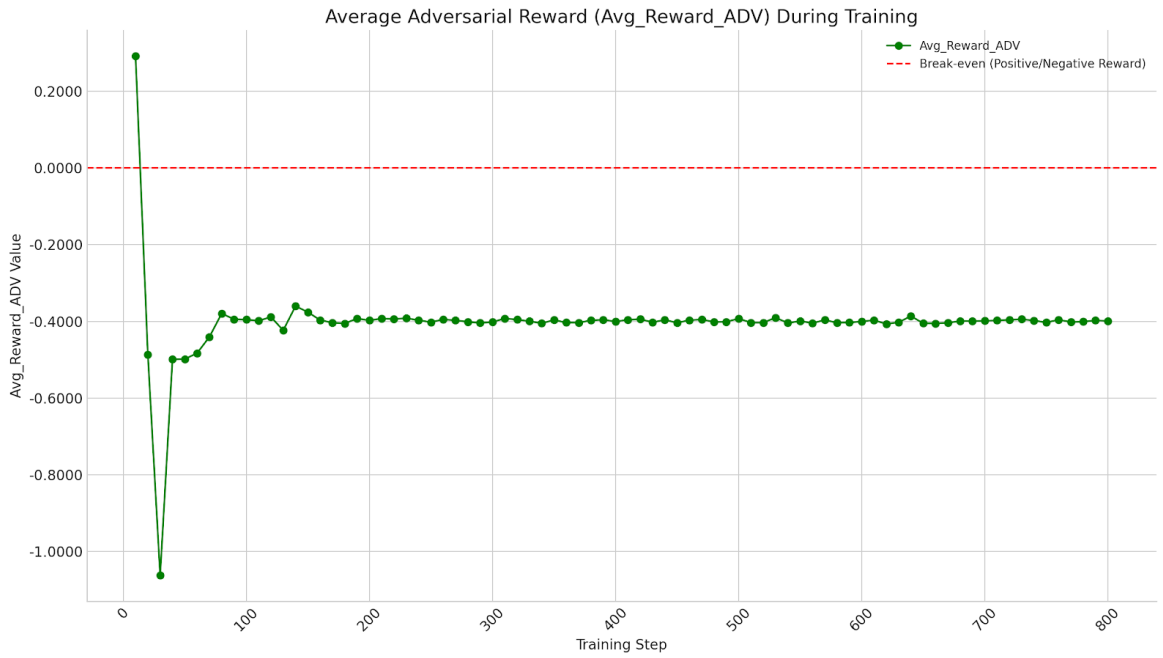
Label Flipped



Final GPT Evaluation

12hours training, 800its
After Adversarial training, the GPT model did not perform better.





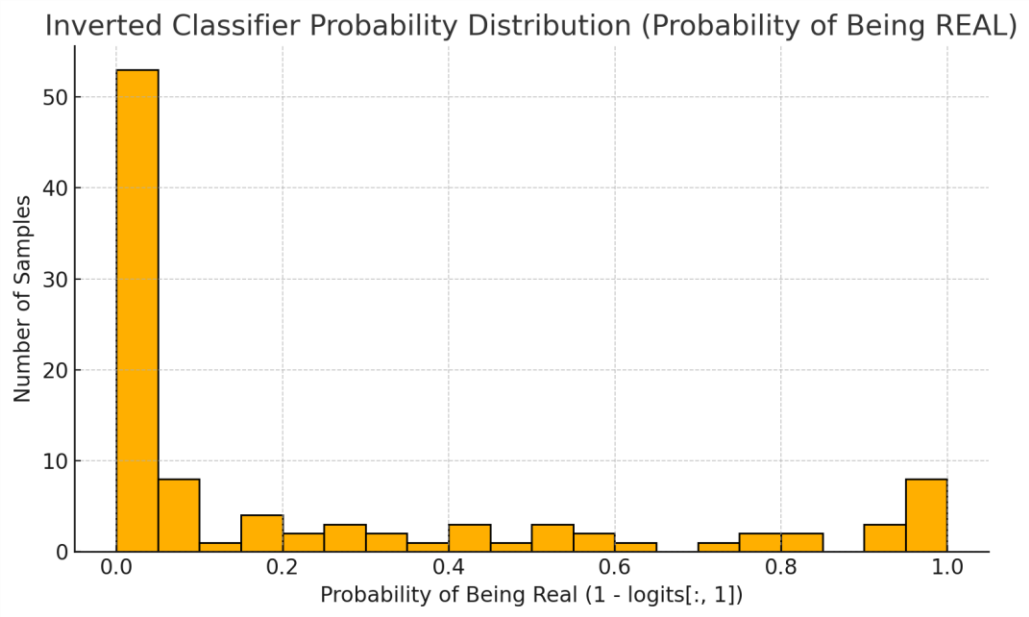
First: Provides the most valuable learning opportunity for the generator

Second: This clearly indicates that the discriminator has completely “won”, the dynamic equilibrium of the training has been broken, and the quality of the generator's output has degraded to “token soup” at this stage.

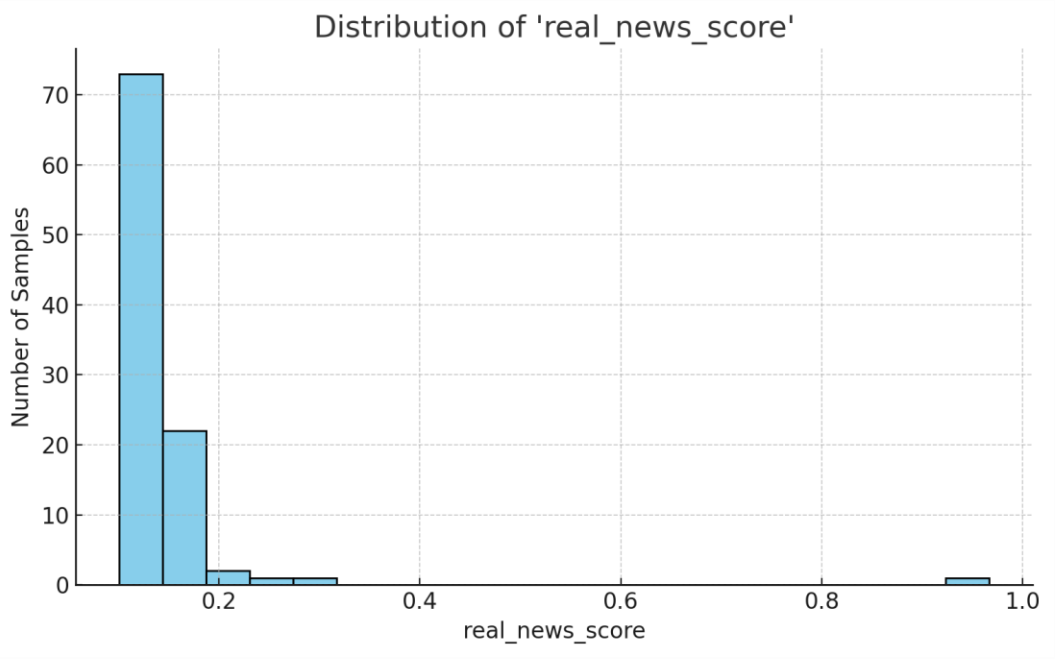
- **Above the line (Positive Reward):** The Generator is winning
- **Below the line (Negative Reward):** The Discriminator is winning

Final Classifier Evaluation

Form the first 70its, we still helped the classifier learn something

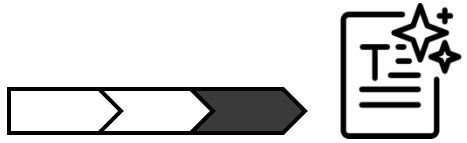


Classifier (before adversarial) performance on 100 news articles generated by GPTv2.



Classifier (after adversarial) performance on 100 news articles generated by GPTv2

Limitation & Further Optimization



- In conclusion, we have trained a relative good GPT-model & Classifier. However, it still has lots of limitations.

- Dataset is small for training a good generator （only 275MB）
- Limited GPU compared to H100

- More Training Loops
- More Advanced Transformer Architecture
- Exploring more robust GAN methods

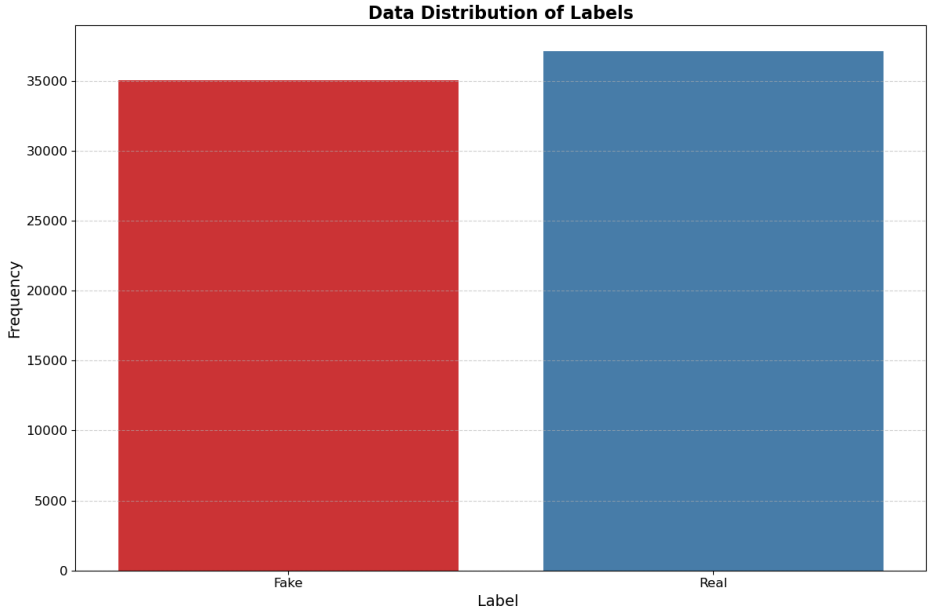
Thank you for your attention!

Appendix

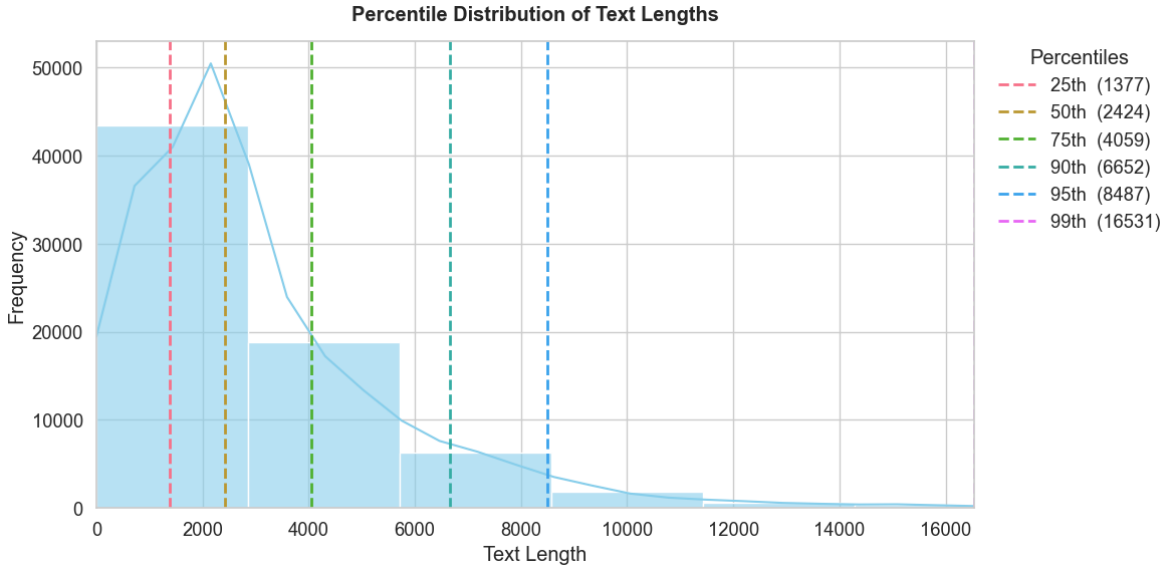
Project Statement

- All participants contributed evenly

APPENDIX: Data Distribution

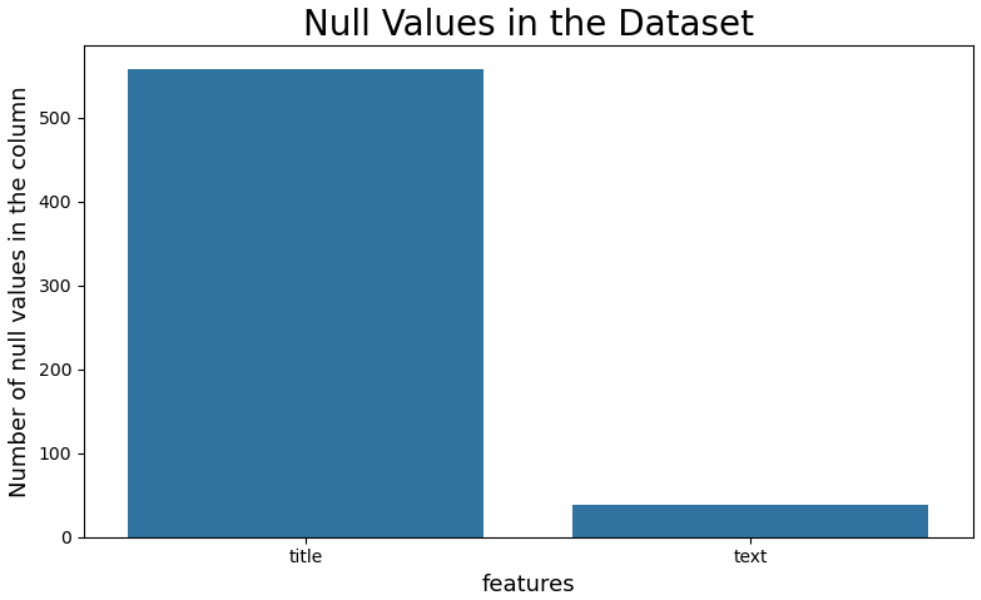


Total Null values distributed in dataset

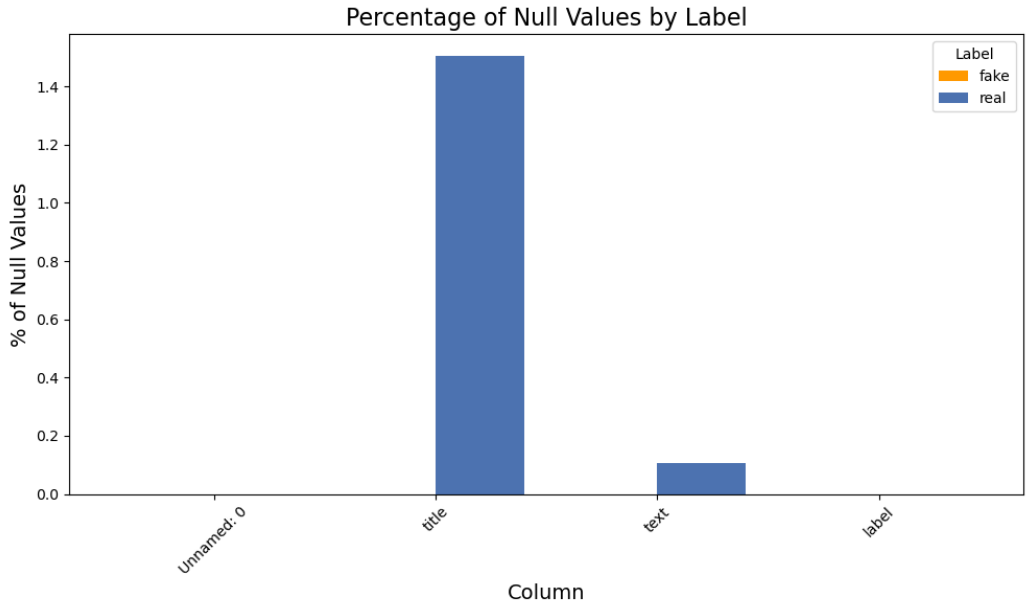


relative Null values distributed in dataset

APPENDIX: Null Statistic

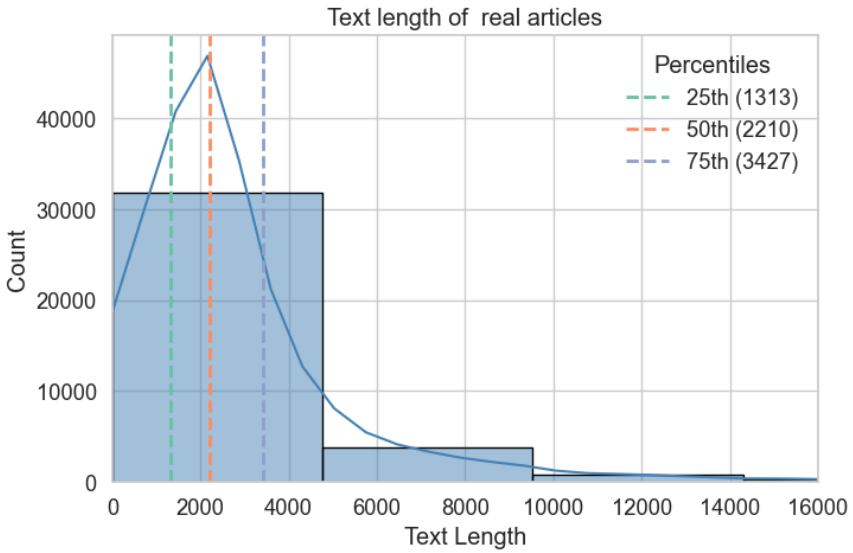


Total Null values distributed in dataset

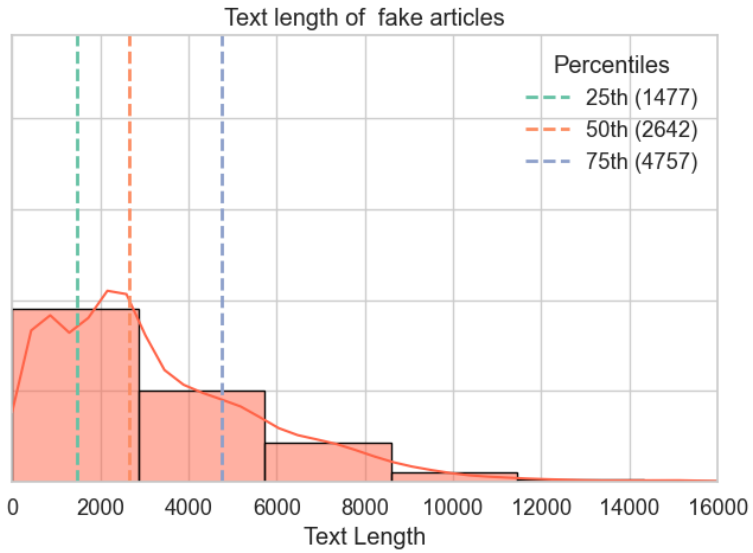


relative Null values distributed in dataset based on label

APPENDIX: Text length based on label

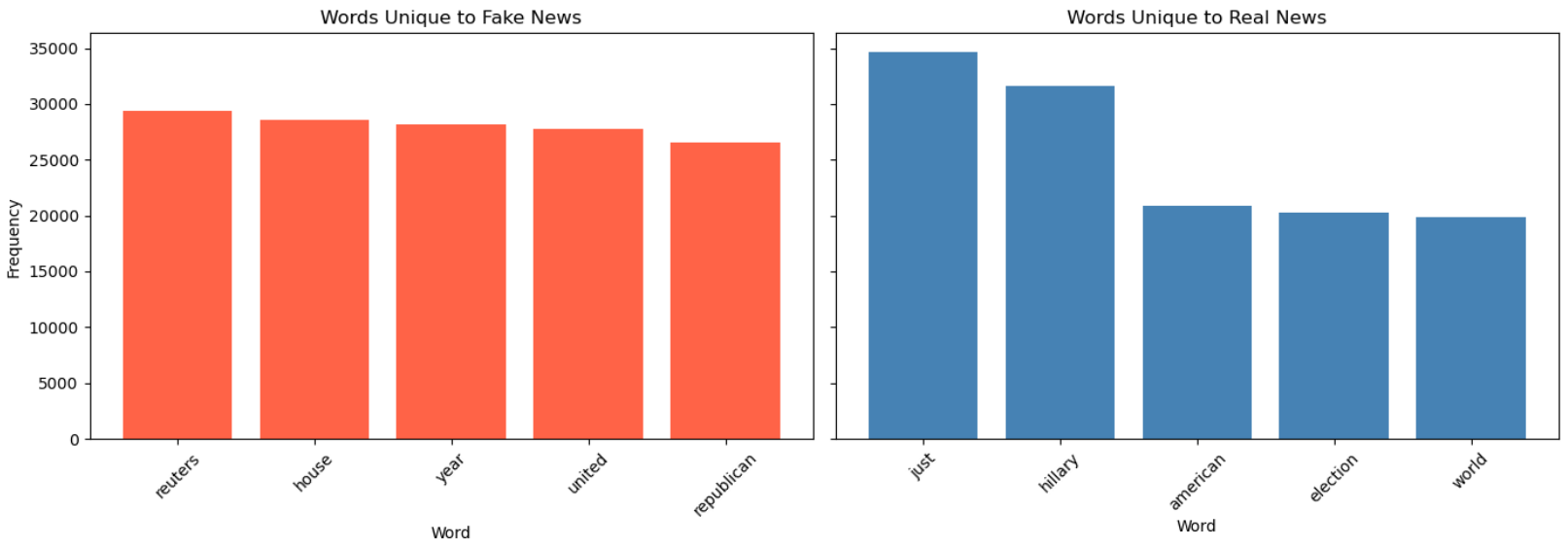


Text length real articles



Text length real articles

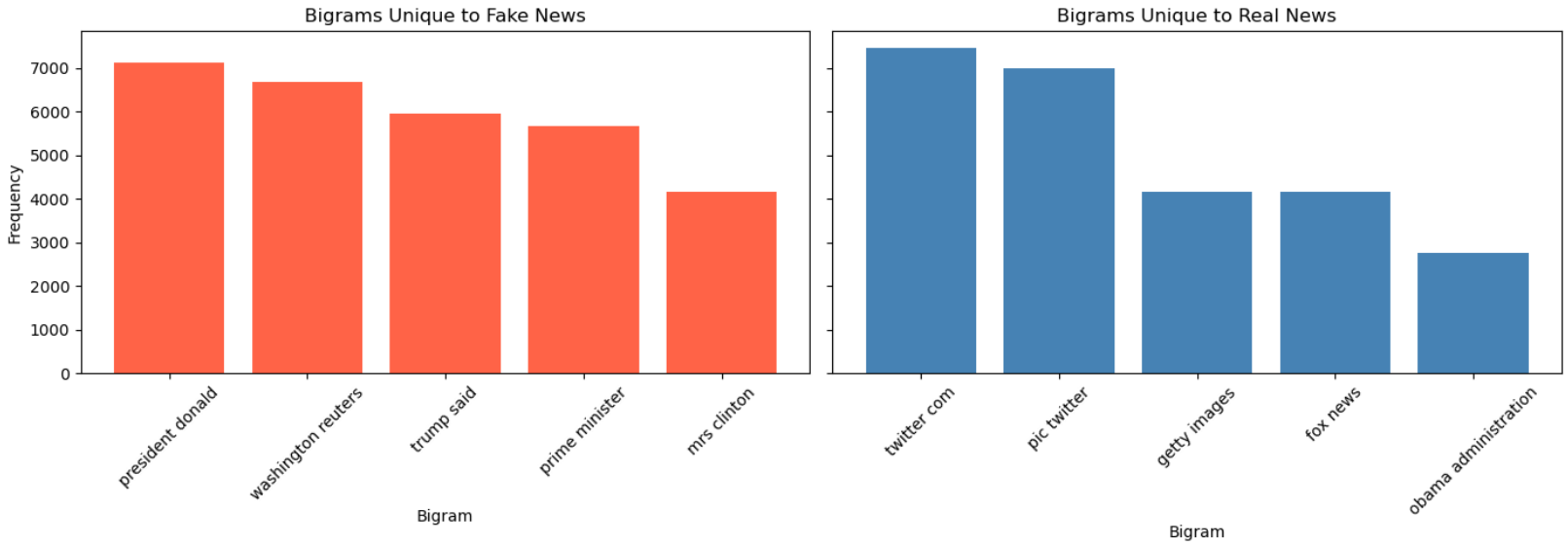
APPENDIX: Frequent Single Words



Fake news

Real news

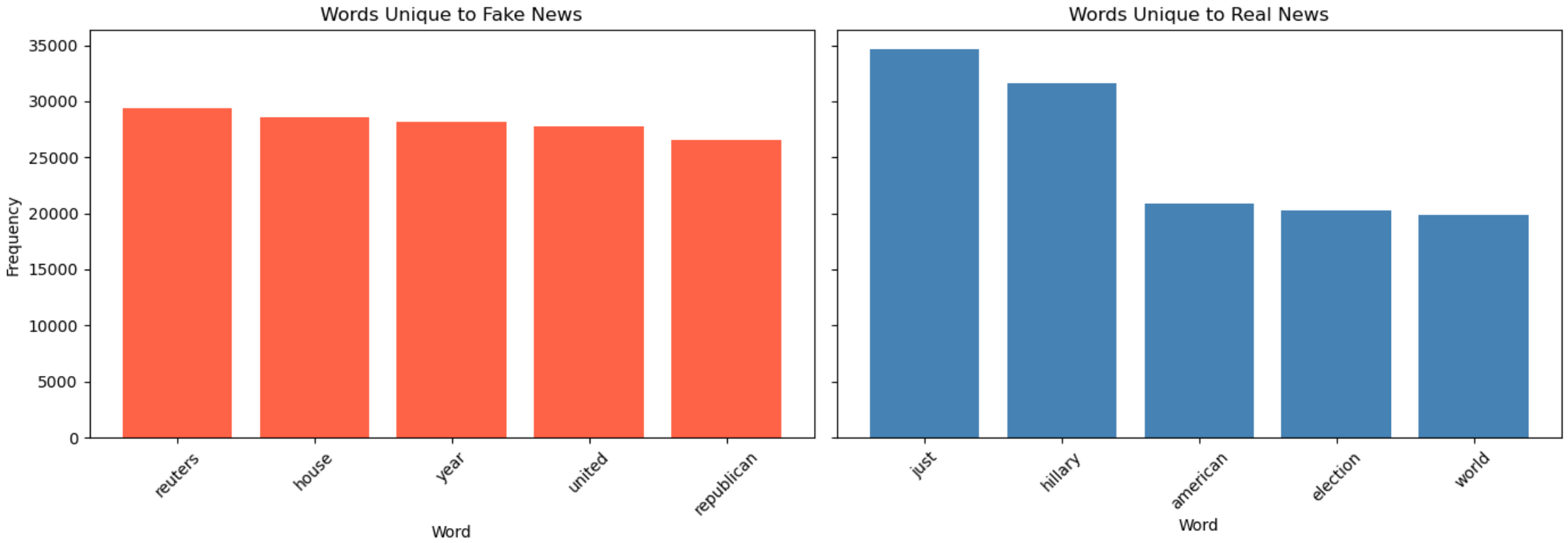
APPENDIX: Frequent Bigrams



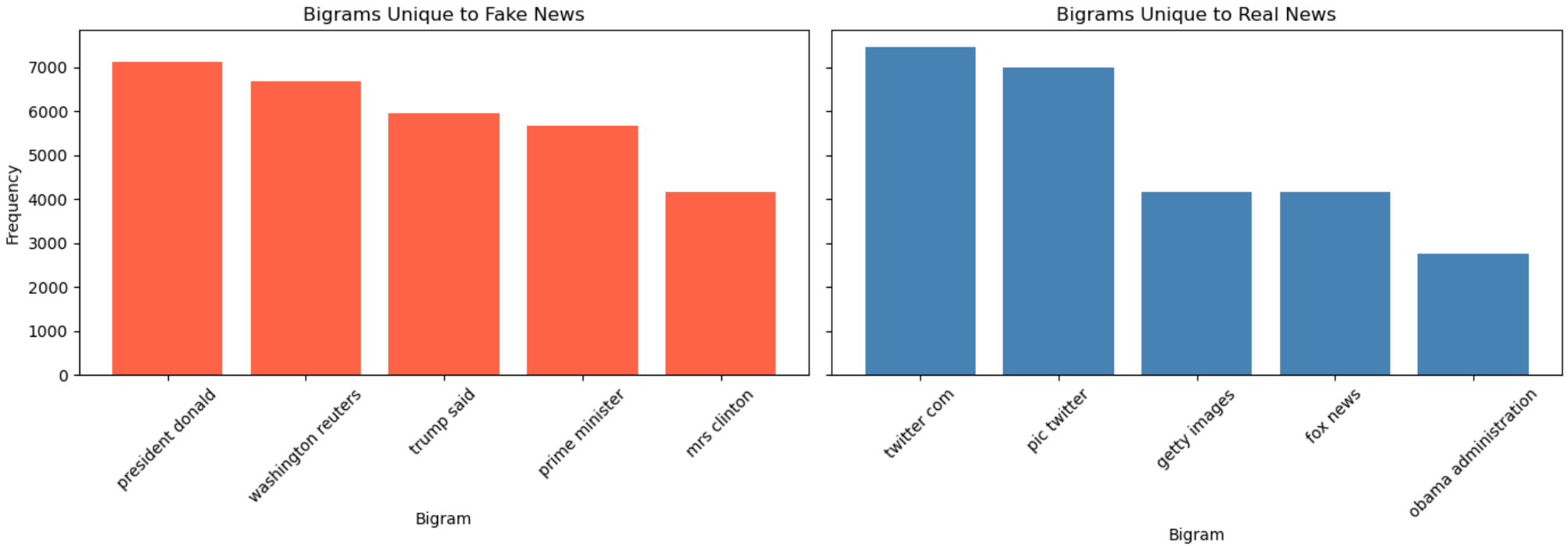
Fake news

Real news

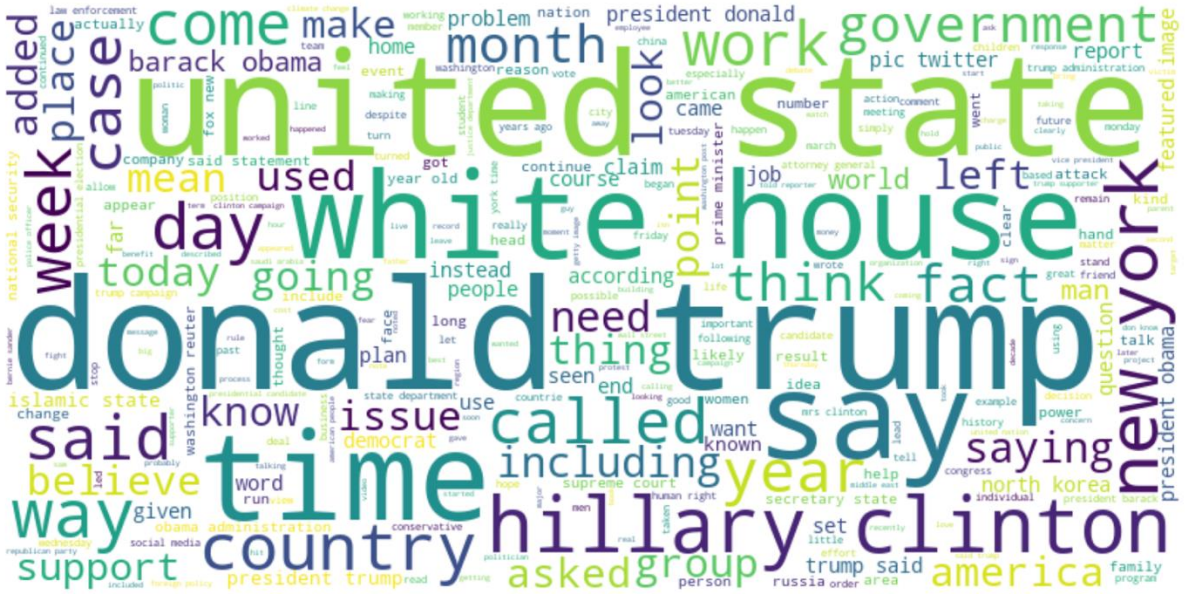
APPENDIX: Top 5 single words unique to top 20 words on fake/real



APPENDIX: Bigrams unique to top 20 words on one fake/real

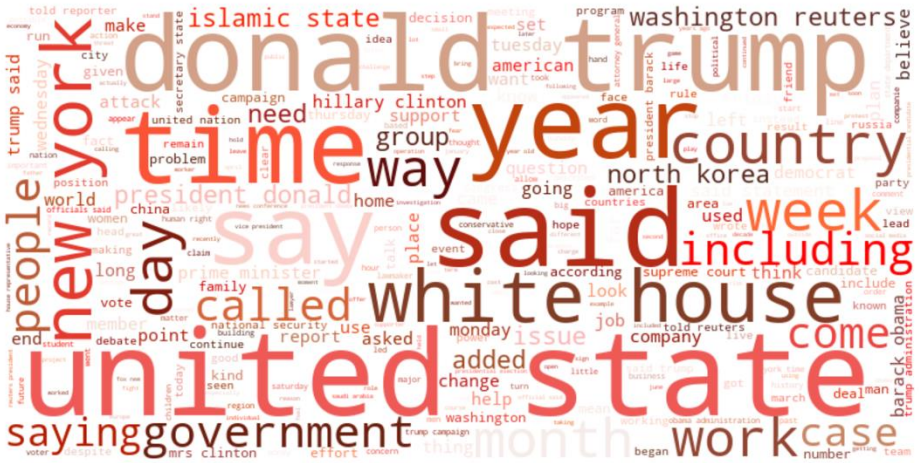


APPENDIX: World Cloud I

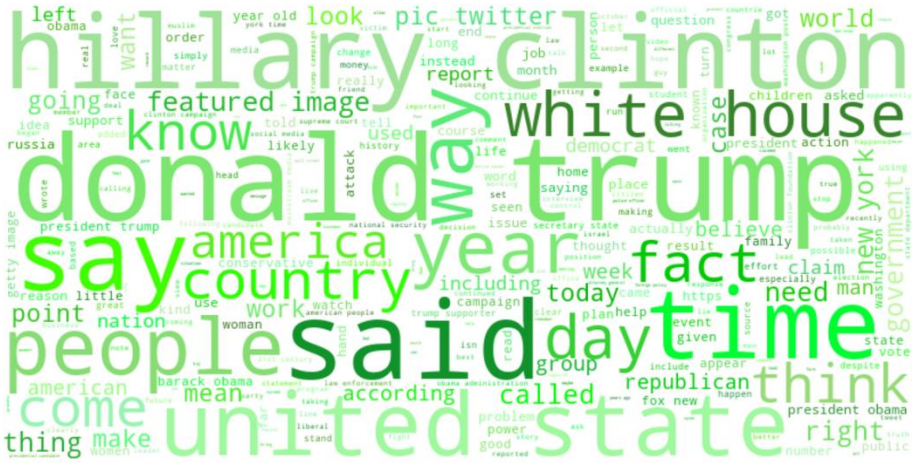


Worldcloud of full dataset

APPENDIX: World Cloud II



Worldcloud of full fake dataset

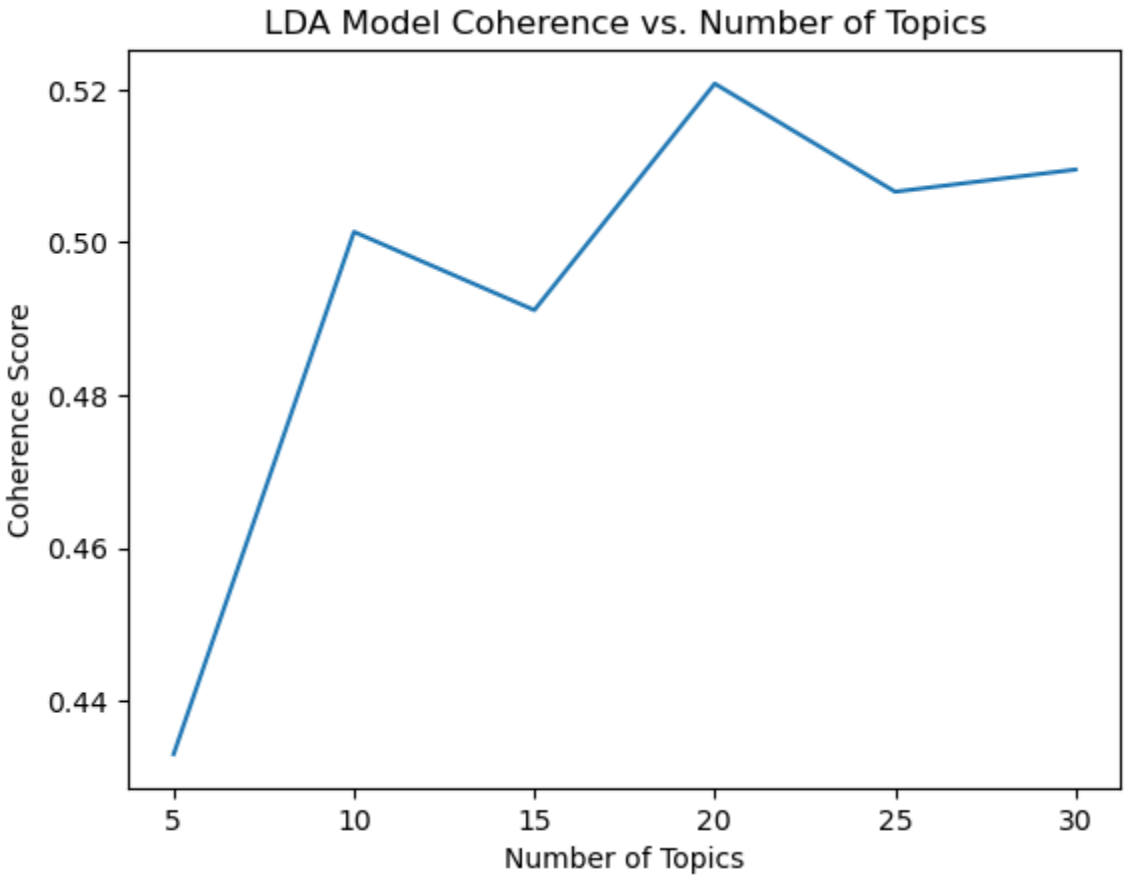


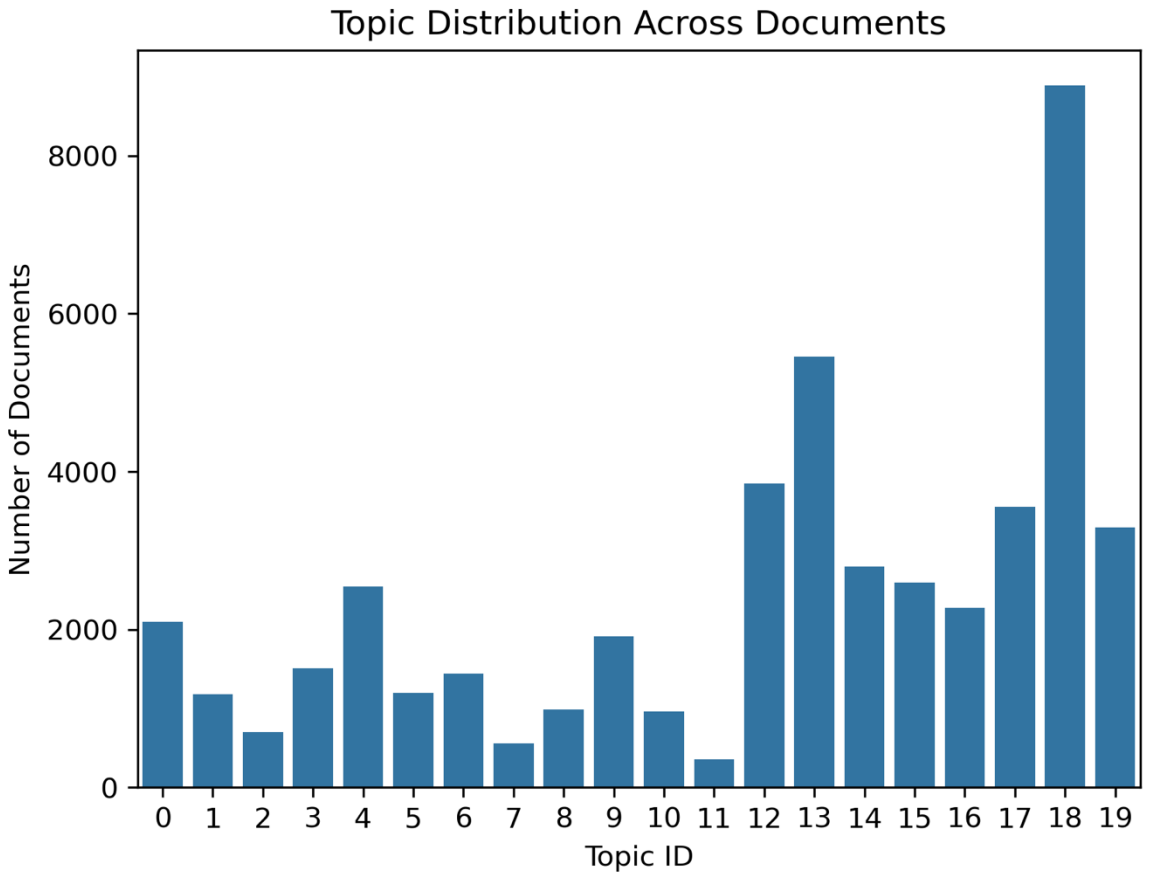
Worldcloud of full real dataset

Topic Modeling with LDA (Preprocessing)

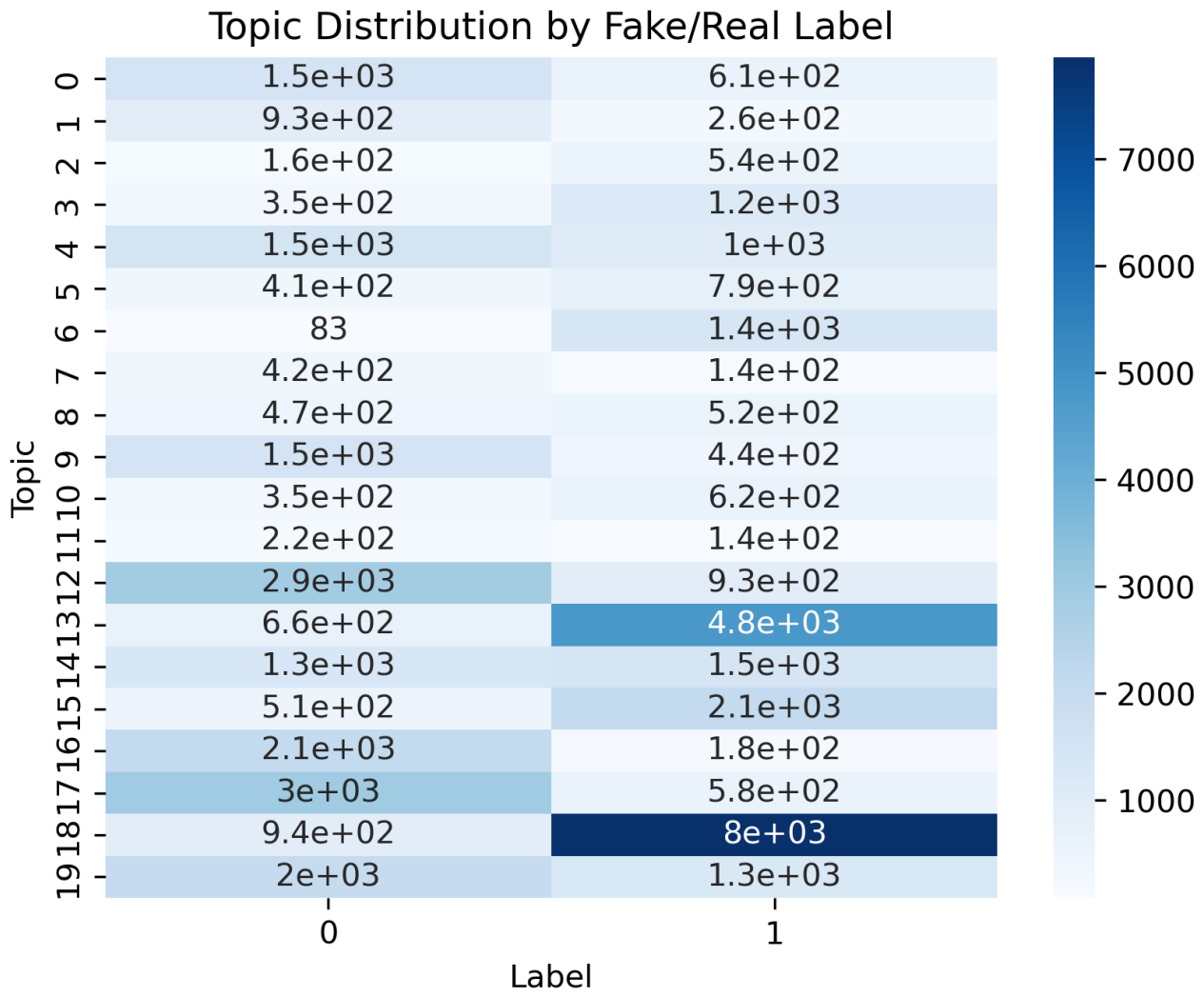
```
from gensim.models.ldamodel import LdaModel
from gensim.models.coherencemodel import CoherenceModel
from gensim.corpora import Dictionary
import matplotlib.pyplot as plt
import pandas as pd
def compute_coherence_values(dictionary, corpus, texts, start=5, limit=50, step=5):
    models = []
    coherence_scores = []
    for num_topics in range(start, limit + 1, step):
        print(f"Training LDA with {num_topics} topics...")
        model = LdaModel(corpus=corpus, id2word=dictionary, num_topics=num_topics, random_state=42, passes=10)
        models.append(model)
        coherencemodel = CoherenceModel(model=model, texts=texts, dictionary=dictionary, coherence='c_v')
        coherence_scores.append(coherencemodel.get_coherence())
    return models, coherence_scores
# Run the comparison
lda_models, coherences = compute_coherence_values(dictionary, corpus, class_data["tokens"], start=5, limit=30, step=5)
# Plot the results
x = list(range(5, 31, 5))
plt.plot(x, coherences)
plt.xlabel("Number of Topics")
plt.ylabel("Coherence Score")
plt.title("LDA Model Coherence vs. Number of Topics")
plt.show()
```

APPENDIX: Topic Modeling with LDA

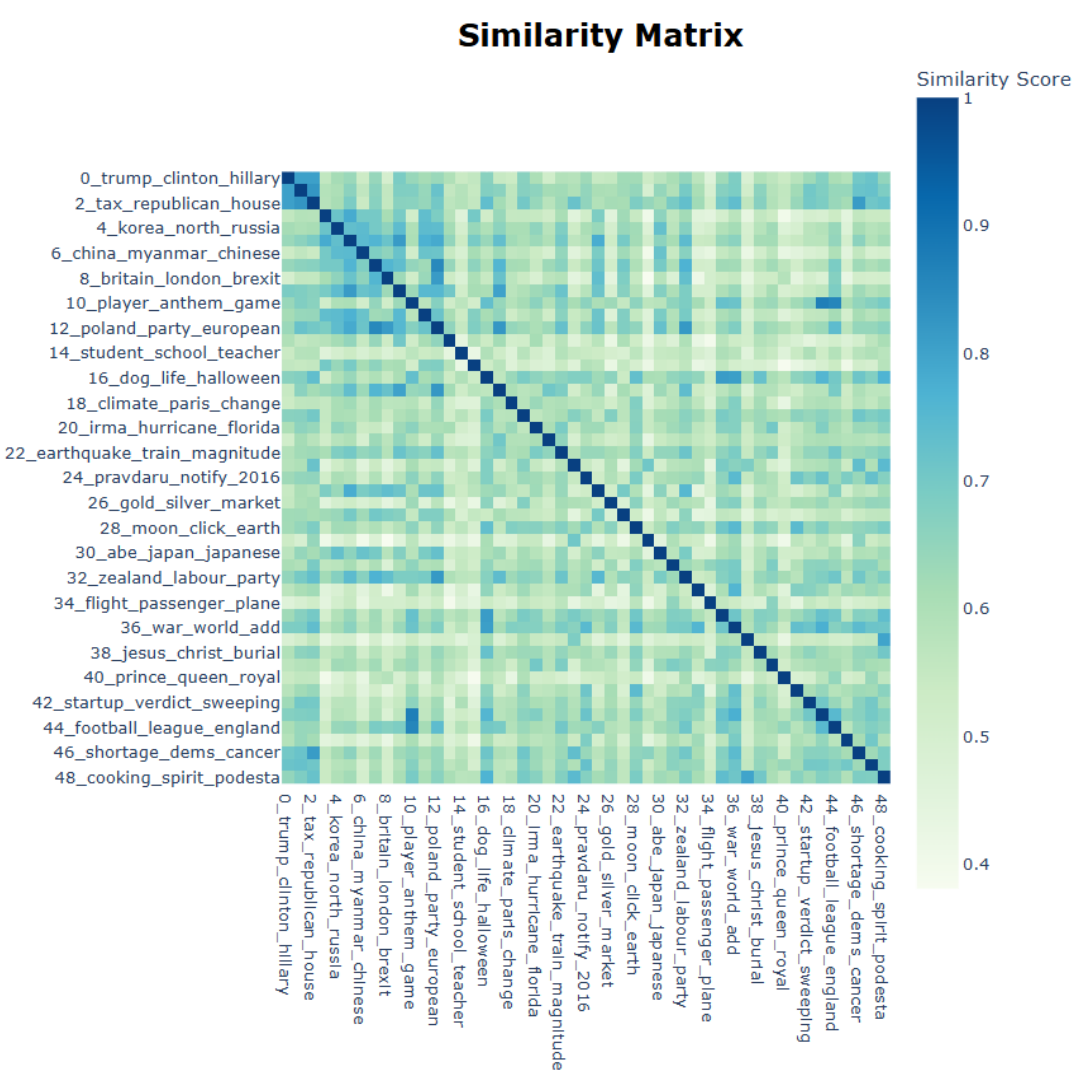


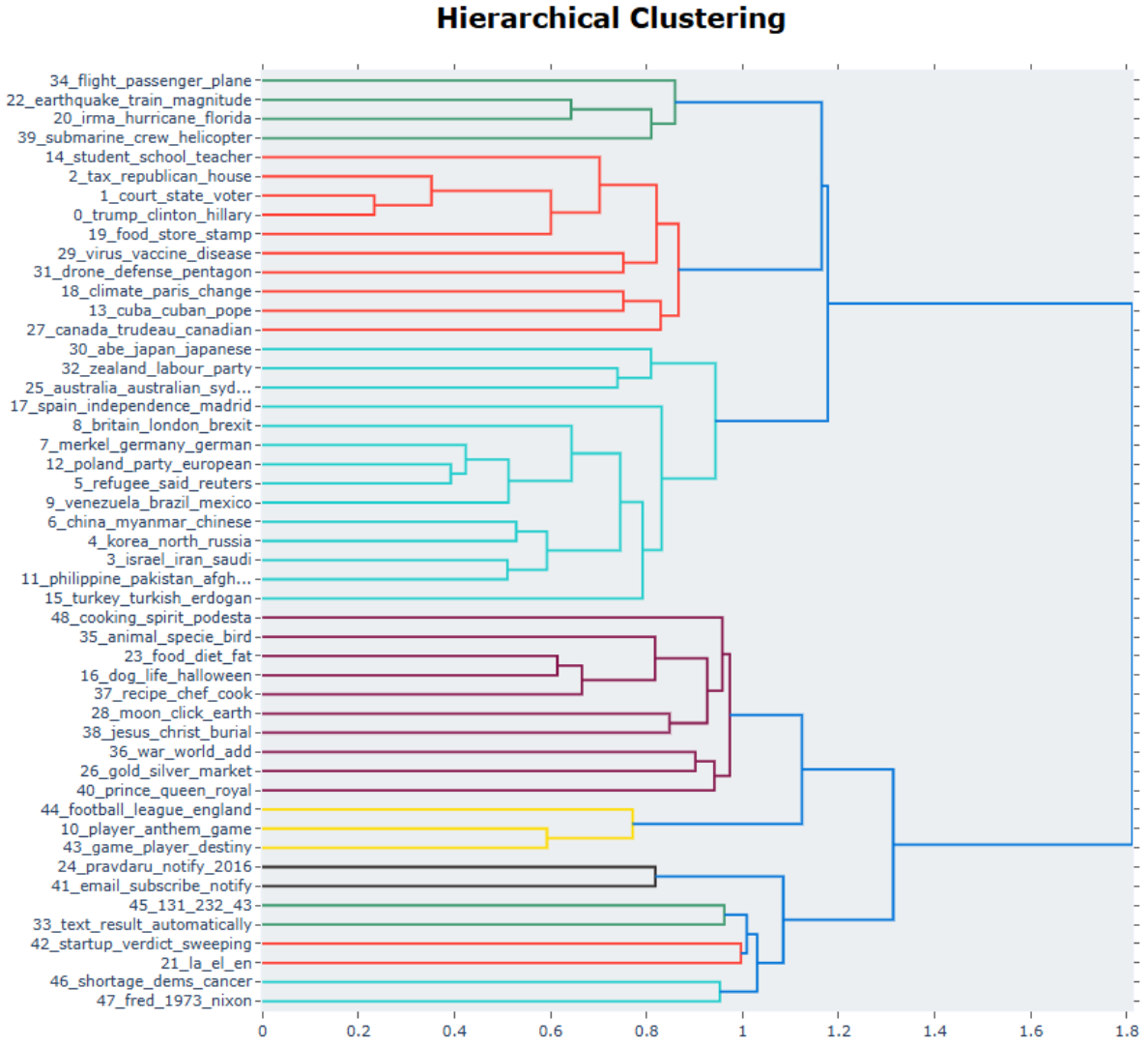


APPENDIX: Topic Modeling with LDA

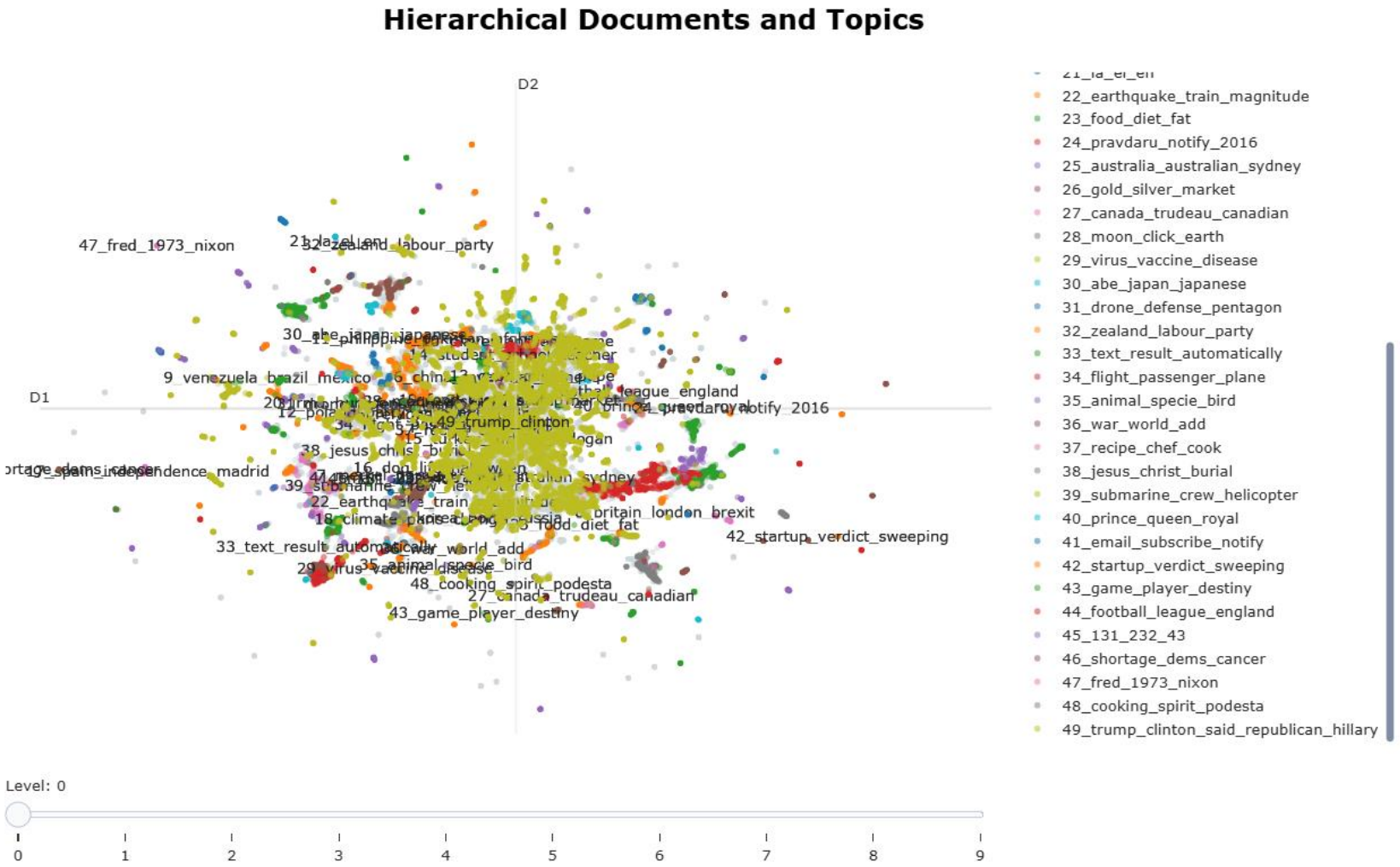


Topic Modeling with BERTopic





Topic Modeling with BERTopic



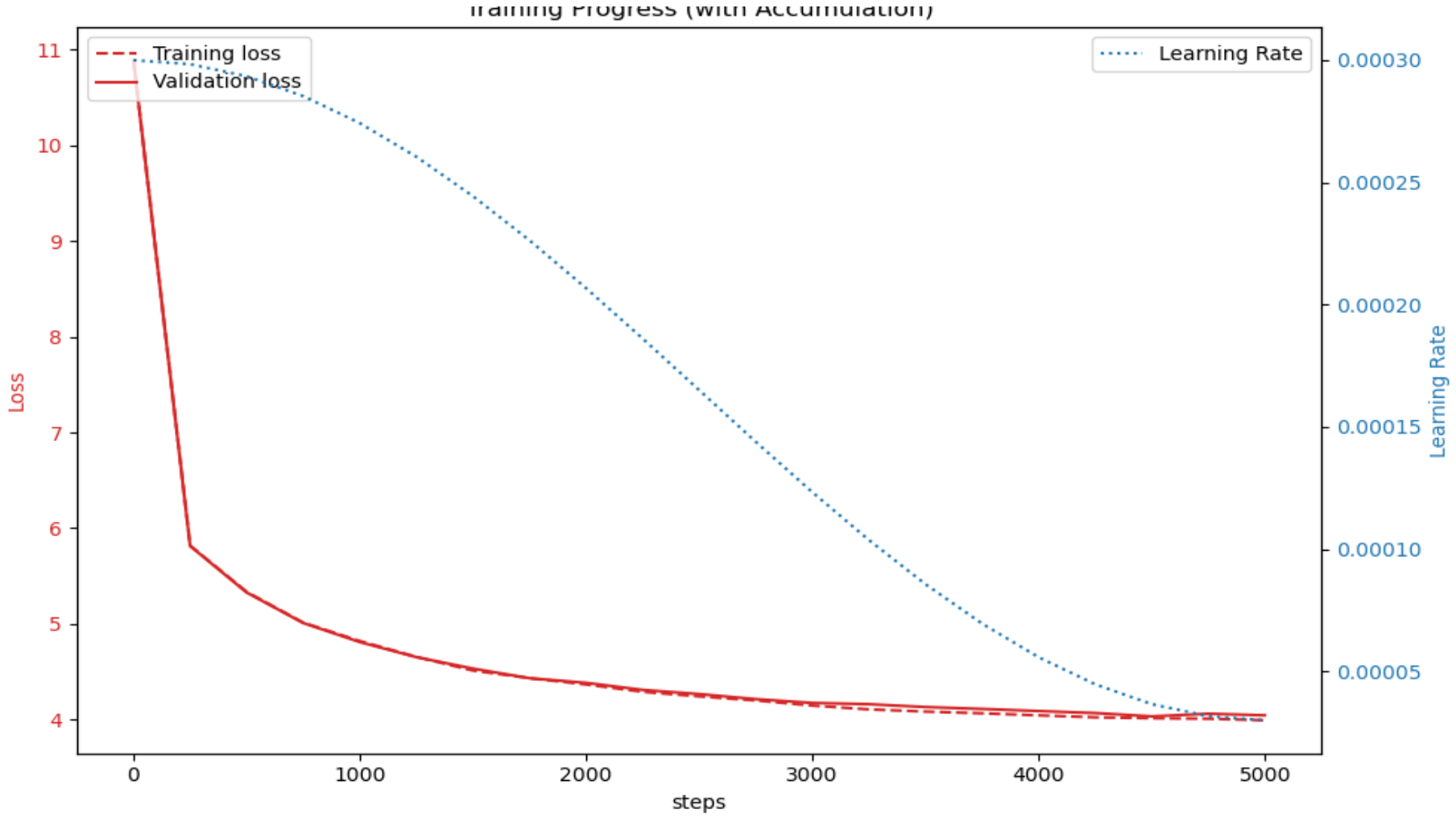
Topic Modeling with BERTopic (Preprocessing)

```
import re
from collections import Counter
nltk.download('stopwords') #remove stopwords
nltk.download('punkt') #remove punctuation
nltk.download('punkt_tab') #needed for further tokenization
nltk.download('wordnet')
nltk.download('omw-1.4')
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
def preprocess_text(text):
    if not isinstance(text, str):
        return ""
    text = text.lower().strip()
    text = re.sub(f"[{re.escape(string.punctuation)}]", "", text)kenize text into words
    tokens = word_tokenize(text) words
    tokens = [t for t in tokens if t not in stop_words]
    tokens = [lemmatizer.lemmatize(t) for t in tokens]
    # Remove rows with less than 7 tokens
    if len(tokens) < 7:
        return ""
    # Join back into string
    return " ".join(tokens)
```

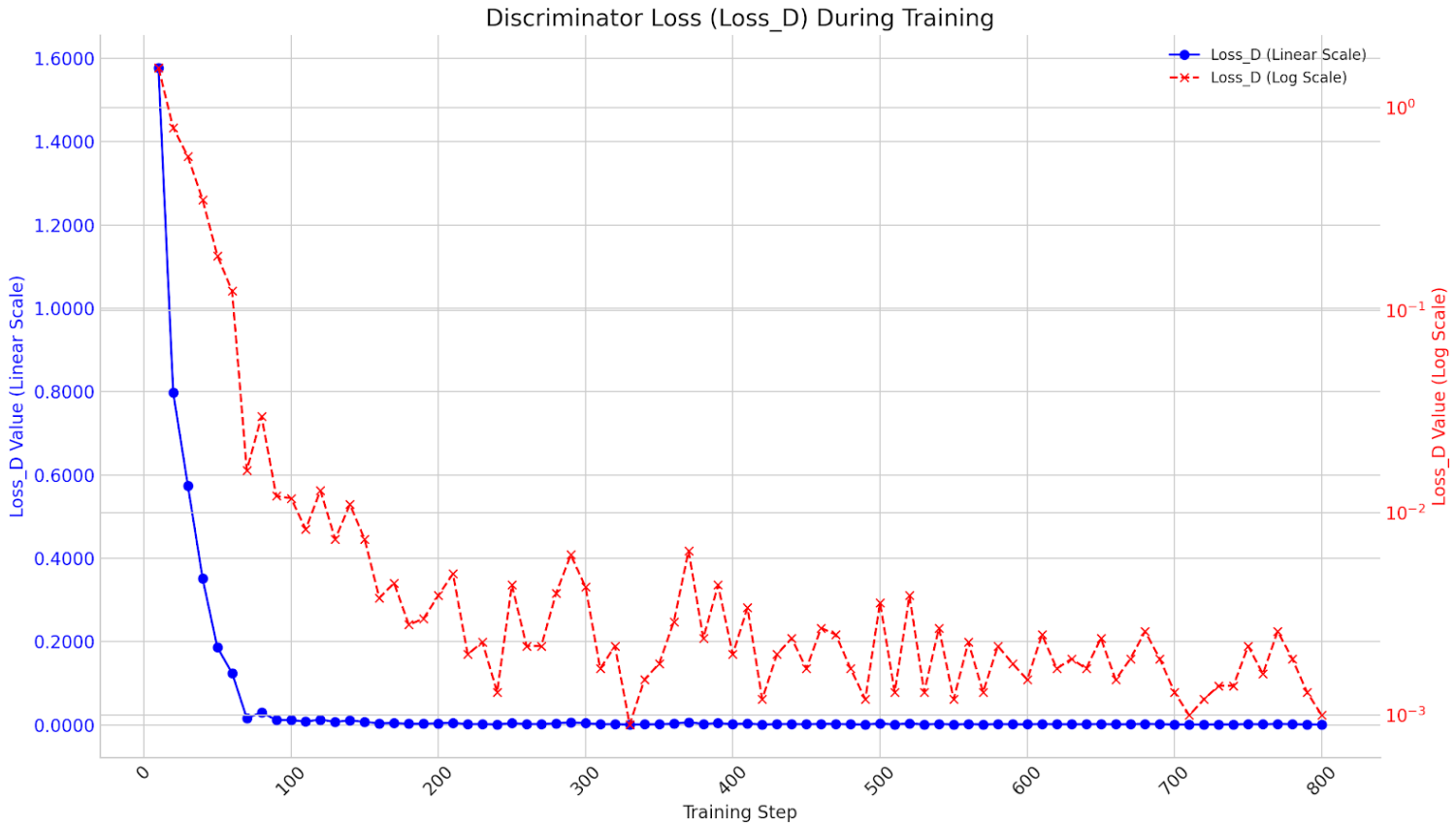
Topic Modeling with BERTopic (Preprocessing)

```
#remove stopwords
class_data['input'] = class_data['input'].apply(preprocess_text)
# Tokenize all texts
class_data['tokens'] = class_data['input'].apply(word_tokenize)
# Remove empty rows after preprocessing
class_data = class_data[class_data['input'].str.strip() != ""]
# Flatten all tokens
all_tokens = [token for tokens in class_data['tokens'] for token in tokens]
token_counts = Counter(all_tokens)
# Create word2idx dictionary (start from 2 to reserve 0 for PAD, 1 for UNK)
n_words = 10000
vocab_size = n_words + 2
vocab = {word: i+2 for i, (word, _) in enumerate(token_counts.most_common())}
vocab['<PAD>'] = 0
vocab['<UNK>'] = 1
def encode_tokens(tokens, vocab):
    return [vocab.get(token, vocab['<UNK>']) for token in tokens]
class_data['encoded'] = class_data['tokens'].apply(lambda tokens: encode_tokens(tokens, vocab))
# Convert to tensors
sequence_tensors = [torch.tensor(seq) for seq in class_data['encoded']]
padded_seqs = pad_sequence(sequence_tensors, batch_first=True, padding_value=vocab['<PAD>'])
max_len = max(len(seq) for seq in sequence_tensors)
min_len = min(len(seq) for seq in sequence_tensors)
print("Min sequence length:", min_len)
print("Max sequence length:", max_len)
#Min sequence length: 7 Max sequence length: 548
```

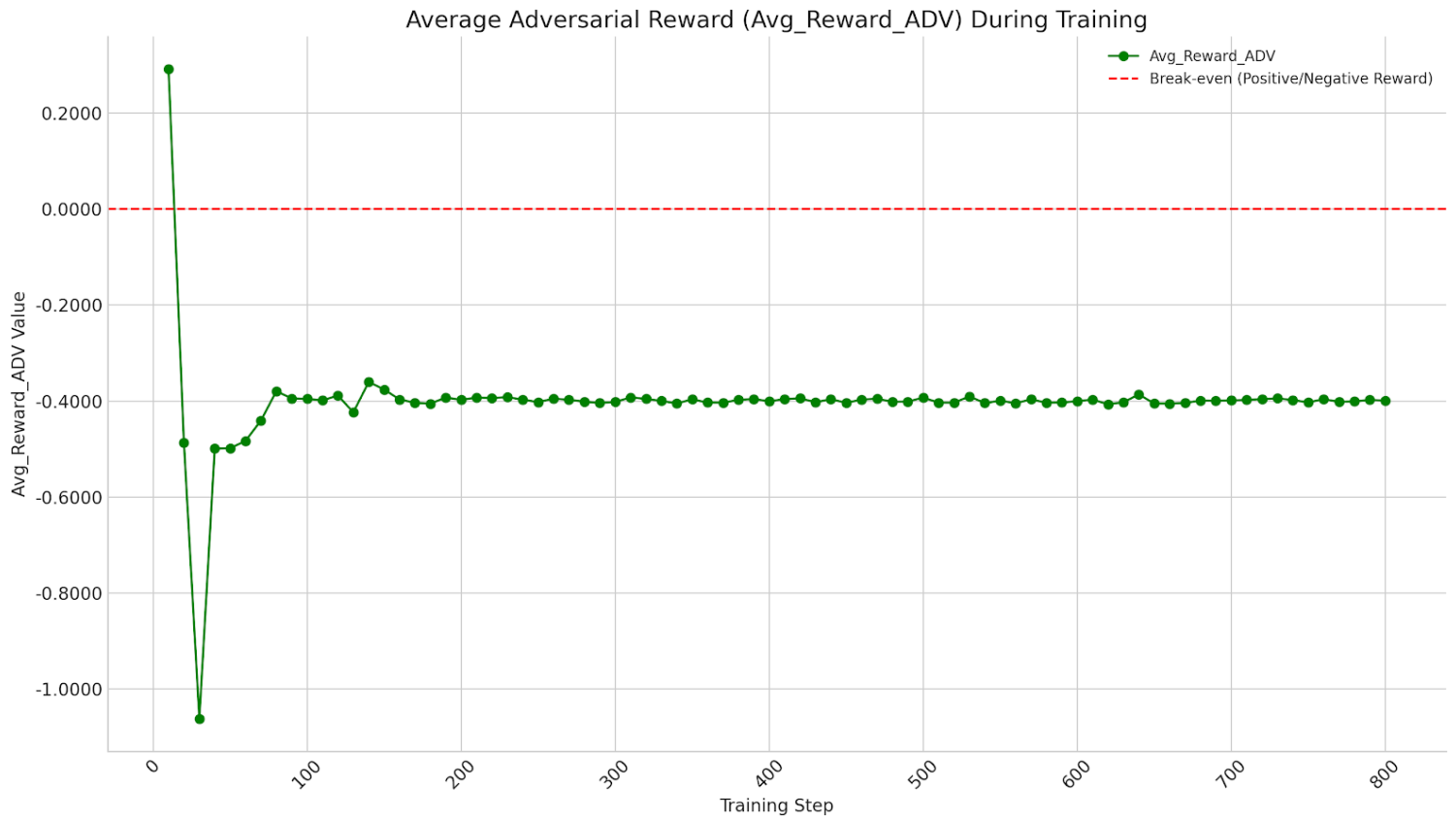
APPENDIX: Training Process



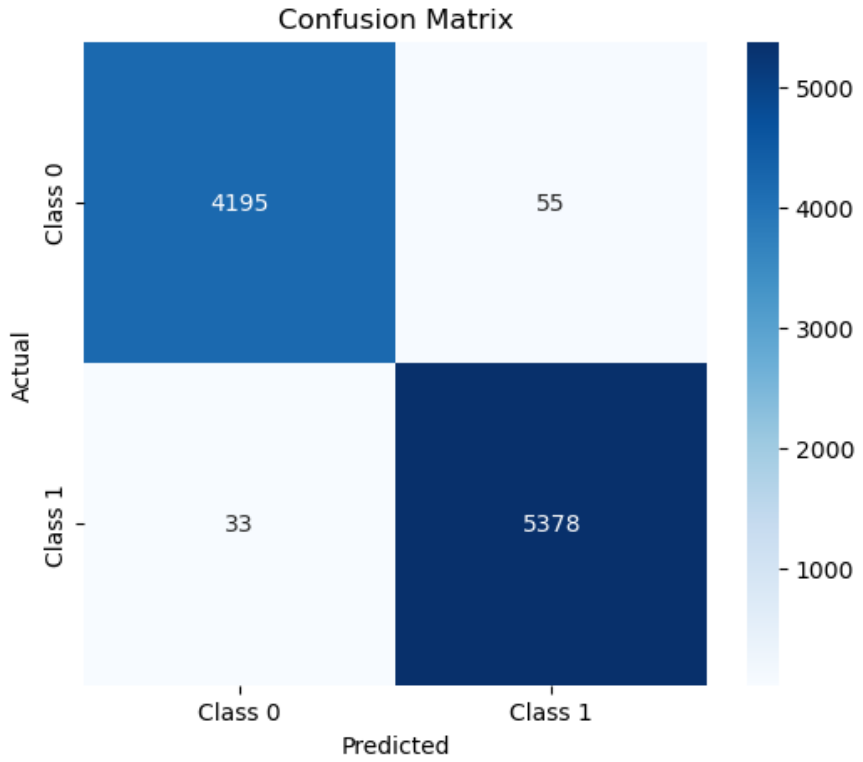
Discriminator Loss in GAN



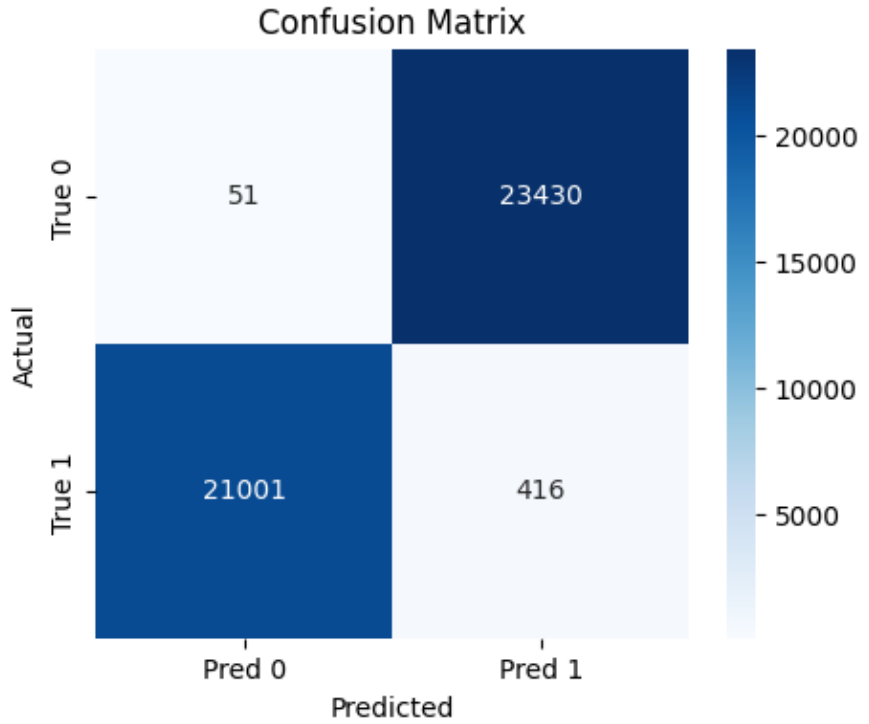
Average Reward in GAN



Confusion Matrix for BERT Classifier

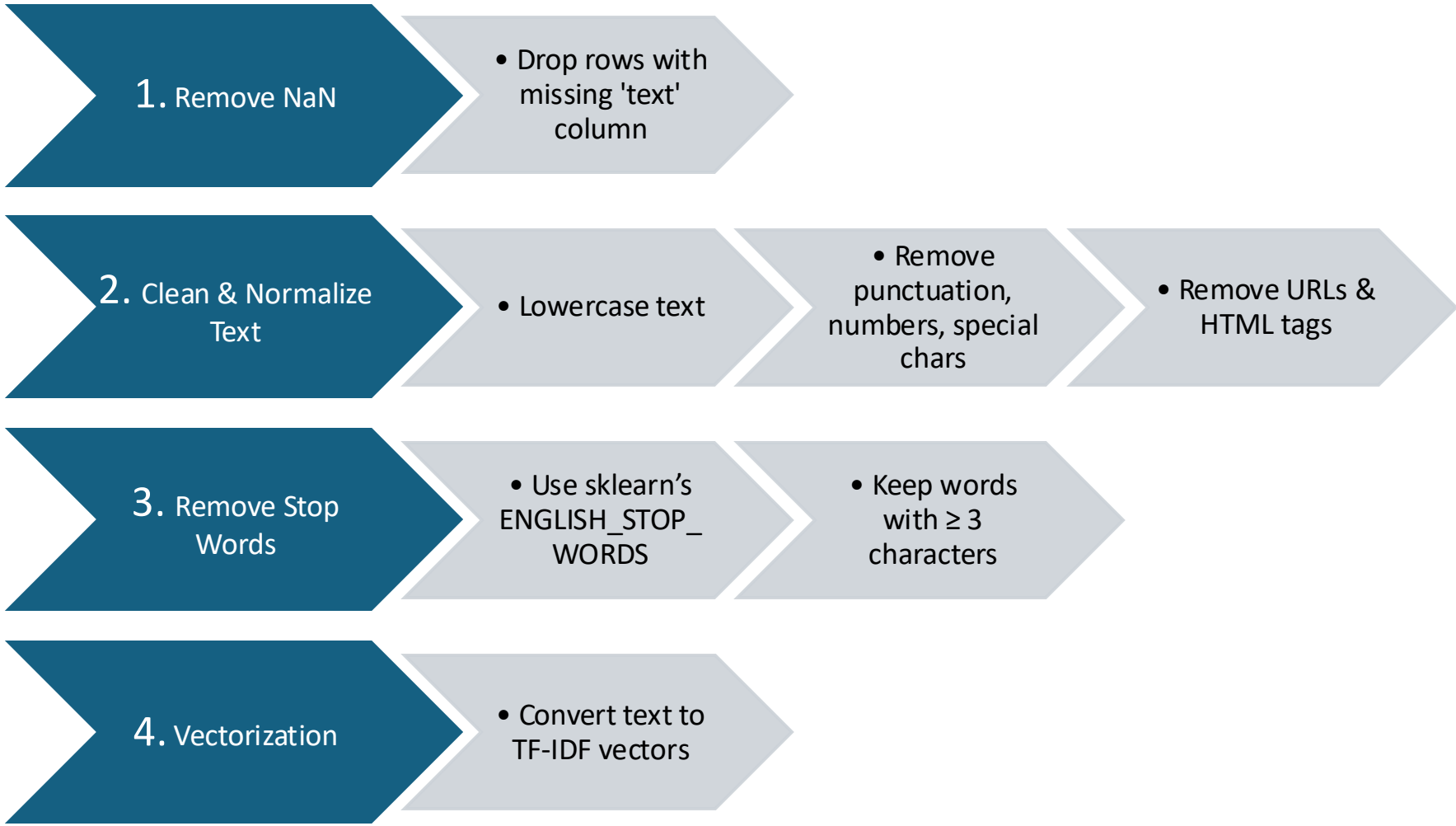


Confusion Matrix for validation set



Confusion Matrix for test set

APPENDIX: Preprocessing flow (non-NN)



APPENDIX: filler removal and text to vector

```python
import re
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS

def remove_fillers(text: str) -> str:

    words = re.findall(r'\b\w+\b', text.lower())
    filtered = [w for w in words if w not in ENGLISH_STOP_WORDS and len(w) >= 3]
    return " ".join(filtered)


data['text'] = data['text'].apply(remove_fillers)
```

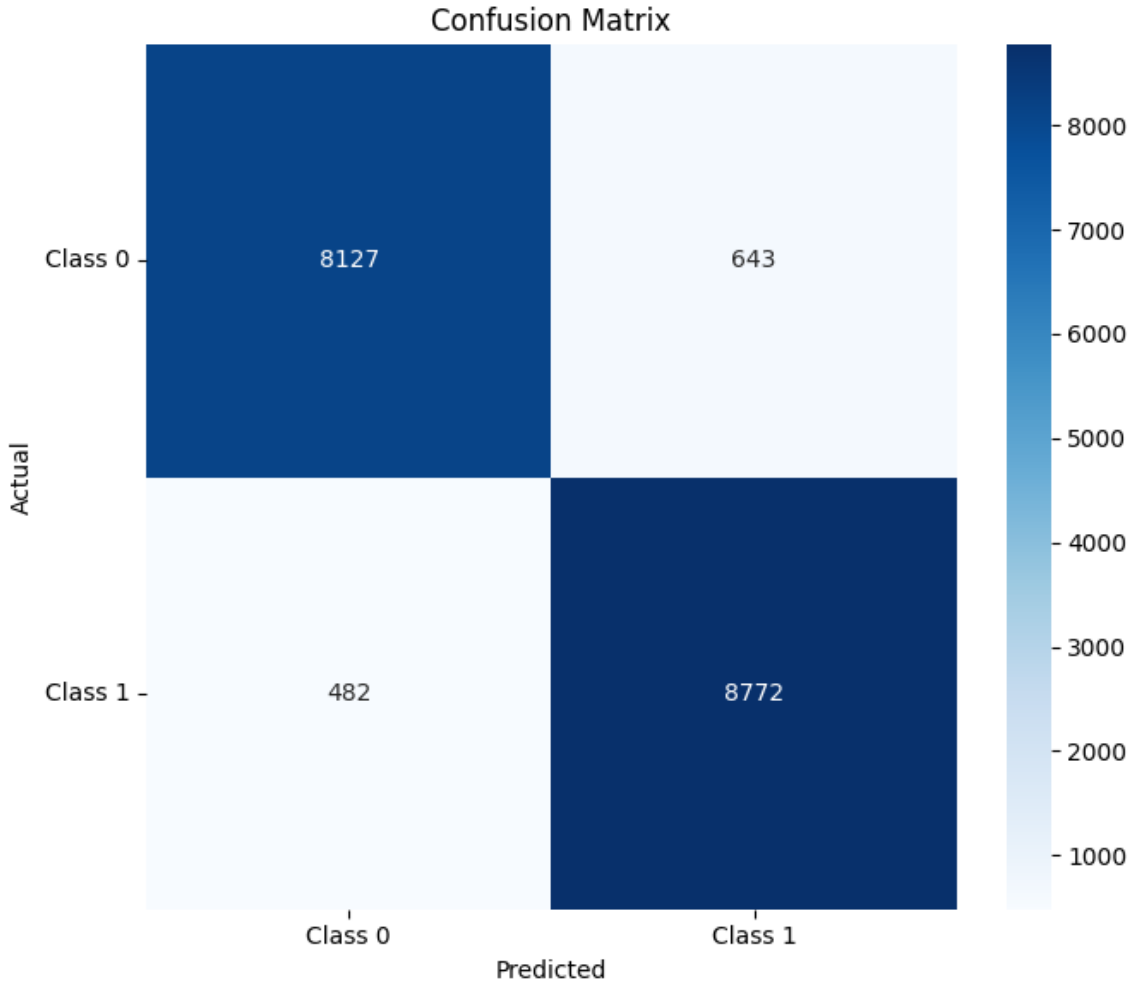
Python

```python
from sklearn.feature_extraction.text import TfidfVectorizer

vectorization = TfidfVectorizer()
xv_train = vectorization.fit_transform(x_train)
xv_test = vectorization.transform(x_test)
```

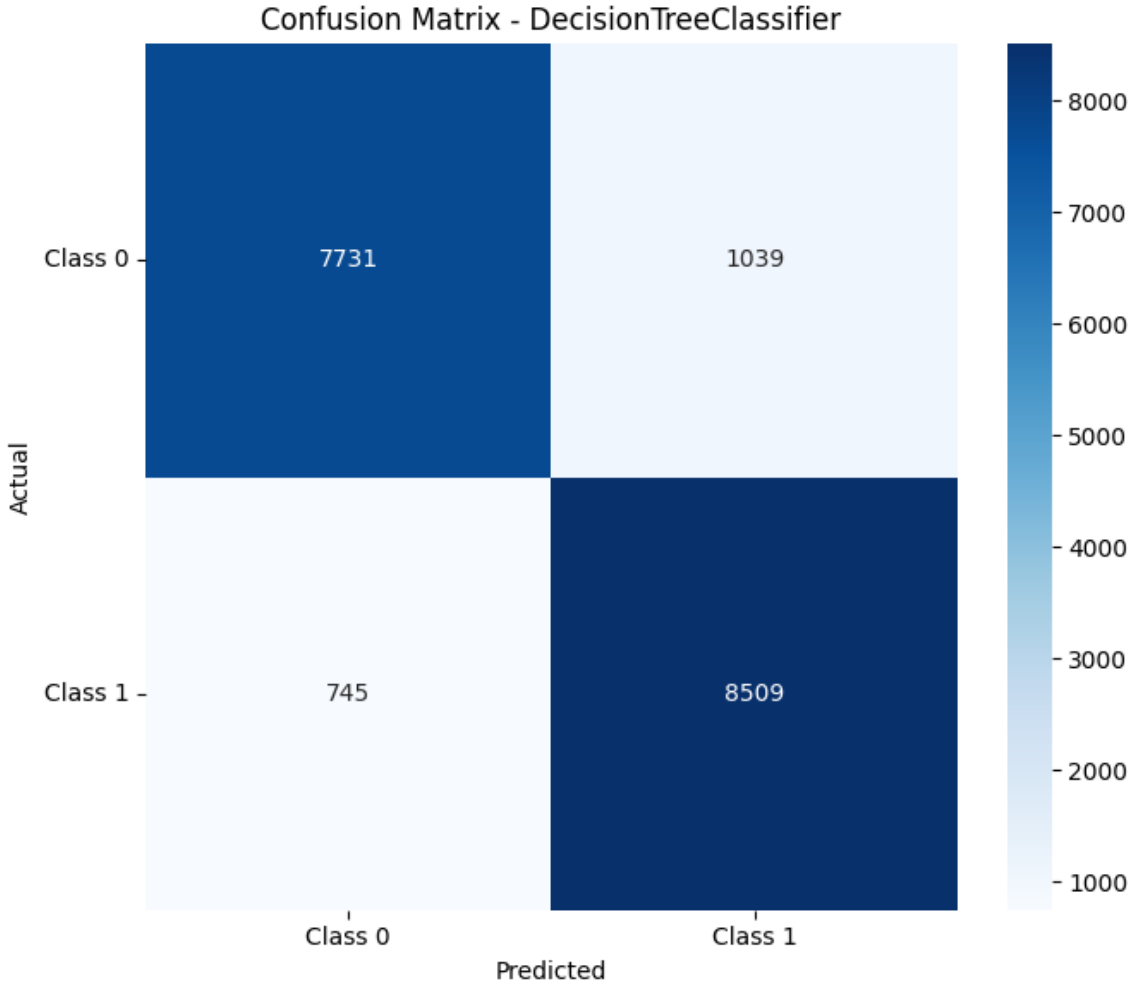
Python

APPENDIX: Logistic regression



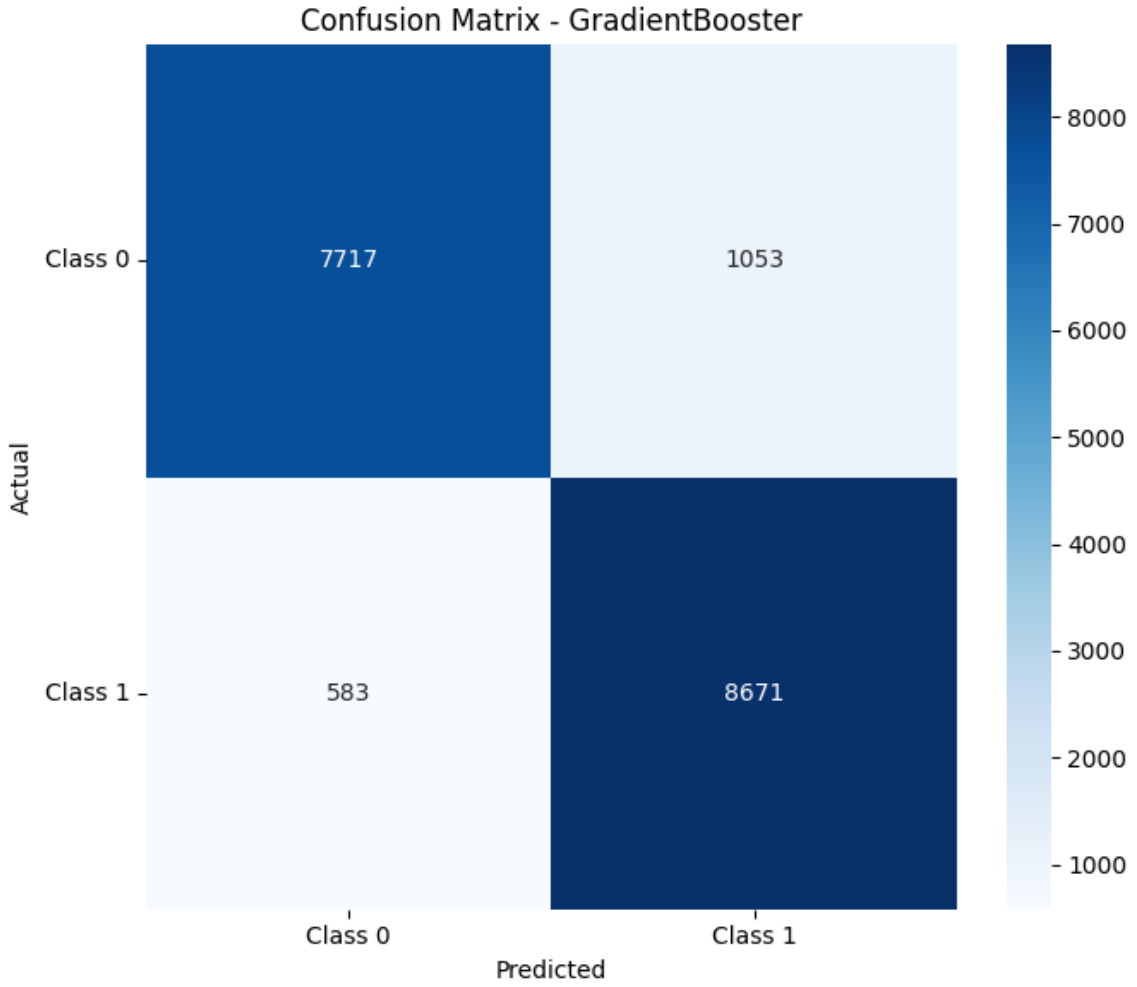
93.7% accuracy

APPENDIX: Decision Tree Classifier



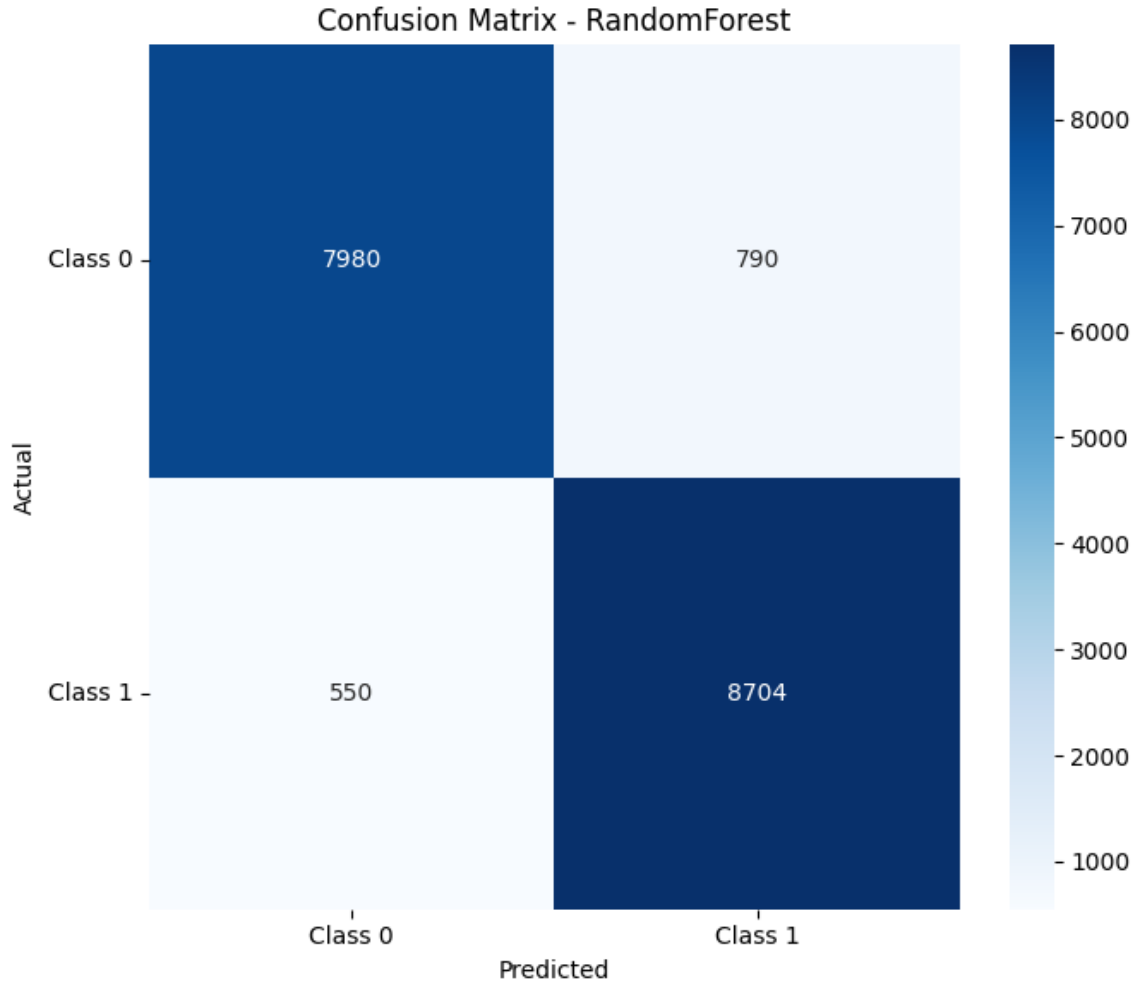
90.1% accuracy

APPENDIX: Gradient Boost Classifier



90.9% accuracy

APPENDIX: Gradient Boost Classifier



92.6% accuracy

APPENDIX: Loading GPT for Adversarial Loop

```
ckpt_path = "gpt_generator_reinforced.pt"
if os.path.exists(ckpt_path):
    print(f" Trying to load the best model from {ckpt_path} ...")
    load_model_config = GPTConfig(
        vocab_size=tok.vocab_size,
        block_size=BLOCK_SIZE,
        n_layer=N_LAYER,
        n_head=N_HEAD,
        n_embd=N_EMBD,
        dropout=DROPOUT,
        pad_token_id=tok.pad_token_id
    )
    load_model = GPT(load_model_config, use_checkpoint=False)
    state_dict = torch.load(ckpt_path, map_location=DEVICE)
    load_model.load_state_dict(state_dict)
    load_model.to(DEVICE)
    load_model.eval()
    model = load_model
    print(f"Form {ckpt_path} loading the best model 。")
else:
    print(f"Failed to load the best model")
```

APPENDIX: Loading Classifier for Adversarial Loop

```
from transformers import BertForSequenceClassification

discriminator = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)
discriminator.load_state_dict(torch.load('best_bert_model.pt', map_location=DEVICE), strict=True)
discriminator.to(DEVICE)
discriminator.eval()
print("Loading the discriminator model from 'best_bert_model.pt'.")
```

```
from transformers import AutoTokenizer


try:
    BERT_TOKENIZER_NAME = discriminator.config._name_or_path
except AttributeError:
    BERT_TOKENIZER_NAME = 'bert-base-uncased' # 或者您知道判别器使用的具体 BERT 模型名称
    print(f"Warning: Cannot form discriminator.config._name_or_path get BERT tokenizer name, using default '{BERT_TOKENIZER_NAME}'.")

print(f" : {BERT_TOKENIZER_NAME}")
bert_tokenizer = AutoTokenizer.from_pretrained(BERT_TOKENIZER_NAME)
print("BERT Tokenizer Successfully Loaded。")
```

APPENDIX: Training loop Classifier on GAN

```python
if model and discriminator and tok and bert_tokenizer and DEVICE and g_optimizer and d_optimizer and REAL_NEWS_CORPUS:
    model.to(DEVICE)
    discriminator.to(DEVICE)


    TOTAL_TRAINING_STEPS = 800

    for step in tqdm(range(1, TOTAL_TRAINING_STEPS + 1), desc="Overall Training Progress"):


        discriminator.train()
        model.eval()

        for _ in range(TRAIN_D_STEPS):
            d_optimizer.zero_grad()

            real_texts_for_d = get_real_text_batch(BATCH_SIZE_FOR_D_REAL_SAMPLES)
            real_encodings_d = bert_tokenizer(real_texts_for_d, padding='max_length', truncation=True, max_length=BERT_MAX_LENGTH, return_tenso
            real_input_ids_d = real_encodings_d.input_ids.to(DEVICE)
            real_attention_mask_d = real_encodings_d.attention_mask.to(DEVICE)
            real_labels_d = torch.ones(real_input_ids_d.size(0), dtype=torch.long, device=DEVICE)

            prompt_text_d_g = "[FAKE] Title: "
            prompt_ids_d_g = tok.encode(prompt_text_d_g, return_tensors="pt").to(DEVICE)

            generated_fake_texts_for_d = []
            current_fake_ids_gpt_batch_list_d = []
            with torch.no_grad():
                for _i in range(BATCH_SIZE_FOR_D_FAKE_SAMPLES):
                    current_fake_ids_gpt = model.generate(
                        prompt_ids_d_g, max_new_tokens=MAX_NEW_TOKENS_GEN,
                        temperature=0.7, top_k=50, eos_token_id=tok.eos_token_id,
                    )
                    current_fake_ids_gpt_batch_list_d.append(current_fake_ids_gpt)
                    decoded_text = tok.decode(current_fake_ids_gpt[0], skip_special_tokens=True, clean_up_tokenization_spaces=True)
                    generated_fake_texts_for_d.append(decoded_text)
```

CONTINUOUS

```
if not generated_fake_texts_for_d:
    if step % PRINT_INTERVAL == 0: print(f"warning ")
    continue

fake_encodings_d = bert_tokenizer(generated_fake_texts_for_d, padding='max_length', truncation=True, max_length=BERT_MAX_LENGTH, ret
fake_input_ids_d = fake_encodings_d.input_ids.to(DEVICE)
fake_attention_mask_d = fake_encodings_d.attention_mask.to(DEVICE)
fake_labels_d = torch.zeros(fake_input_ids_d.size(0), dtype=torch.long, device=DEVICE)

d_all_input_ids = torch.cat([real_input_ids_d, fake_input_ids_d], dim=0)
d_all_attention_masks = torch.cat([real_attention_mask_d, fake_attention_mask_d], dim=0)
d_all_labels = torch.cat([real_labels_d, fake_labels_d], dim=0)

outputs_d = discriminator(input_ids=d_all_input_ids, attention_mask=d_all_attention_masks, labels=d_all_labels)
loss_d = outputs_d.loss
loss_d.backward()
torch.nn.utils.clip_grad_norm_(discriminator.parameters(), GRADIENT_CLIP_VAL)
d_optimizer.step()
```

APPENDIX: Training loop GPT on GAN

```
model.train()
discriminator.eval()

for _ in range(TRAIN_G_STEPS):
    g_optimizer.zero_grad()
    total_loss_g = torch.tensor(0.0, device=DEVICE)
    loss_lm_val, loss_adv_val, reward_adv_val = None, None, None

    if ALPHA_LM > 0:
        real_texts_for_lm = get_real_text_batch(BATCH_SIZE_LM)
        lm_encodings = tok(real_texts_for_lm, padding='max_length', truncation=True, max_length=BLOCK_SIZE, return_tensors='pt')
        lm_input_ids = lm_encodings.input_ids.to(DEVICE)
        lm_targets = lm_input_ids.clone()
        logits_lm, loss_lm_internal = model(lm_input_ids, targets=lm_targets)
        if loss_lm_internal is not None:
            total_loss_g += ALPHA_LM * loss_lm_internal
            loss_lm_val = loss_lm_internal.item()
        else:
            pad_idx = tok.pad_token_id if tok.pad_token_id is not None else -100
            loss_lm_manual = F.cross_entropy(logits_lm.reshape(-1, logits_lm.size(-1)), lm_targets.reshape(-1), ignore_index=pad_idx)
            total_loss_g += ALPHA_LM * loss_lm_manual
            loss_lm_val = loss_lm_manual.item()

    if BETA_ADV > 0:
        prompt_text_g_adv = "[FAKE] Title: "
        prompt_ids_g_adv = tok.encode(prompt_text_g_adv, return_tensors="pt").to(DEVICE)
        prompt_len_adv = prompt_ids_g_adv.size(1)

        adv_fake_ids_gpt_list = []
        adv_generated_texts = []
        for _i in range(BATCH_SIZE_ADV):
            with torch.no_grad():
                current_adv_fake_ids_gpt = model.generate(
                    prompt_ids_g_adv, max_new_tokens=MAX_NEW_TOKENS_GEN,
                    temperature=0.7, top_k=50, eos_token_id=tok.eos_token_id,
                )
            adv_fake_ids_gpt_list.append(current_adv_fake_ids_gpt)

            decoded_text = tok.decode(current_adv_fake_ids_gpt[0], skip_special_tokens=True, clean_up_tokenization_spaces=True)
            adv_generated_texts.append(decoded_text)
```

APPENDIX: CONTINUOUS

```
if not adv_fake_ids_gpt_list:
    if step % PRINT_INTERVAL == 0: print(f"Warning(G_ADV)))")
else:

    max_len_in_adv_batch = 0
    for ids_tensor in adv_fake_ids_gpt_list:
        if ids_tensor.size(1) > max_len_in_adv_batch:
            max_len_in_adv_batch = ids_tensor.size(1)

    padded_adv_fake_ids_gpt_list = []

    pad_token_to_use_gpt = tok.pad_token_id if tok.pad_token_id is not None and tok.pad_token_id < len(tok) else \
                           (tok.eos_token_id if tok.eos_token_id is not None and tok.eos_token_id < len(tok) else 0)
    if pad_token_to_use_gpt >= len(tok):
        print(f"Fatal Error(G_ADV): pad_token_to_use_gpt ({pad_token_to_use_gpt}) out of bounds for GPT tokenizer vocab size ({


    for ids_tensor in adv_fake_ids_gpt_list:
        num_padding = max_len_in_adv_batch - ids_tensor.size(1)
        if num_padding > 0:
            padding_tensor = torch.full(
                (ids_tensor.size(0), num_padding),
                pad_token_to_use_gpt,
                dtype=ids_tensor.dtype,
                device=ids_tensor.device
            )
            padded_tensor = torch.cat([ids_tensor, padding_tensor], dim=1)
            padded_adv_fake_ids_gpt_list.append(padded_tensor)
        else:
            padded_adv_fake_ids_gpt_list.append(ids_tensor)

    adv_fake_ids_gpt_batch = torch.cat(padded_adv_fake_ids_gpt_list, dim=0)
    if adv_fake_ids_gpt_batch.size(0) != BATCH_SIZE_ADV:
    if step % PRINT_INTERVAL == 0 : print(f"   {step} - G_ADV - adv_fake_ids_gpt_batch device: {adv_fake_ids_gpt_batch.device},
```

APPENDIX:CONTINUOUS

```
adv_fake_ids_gpt_batch = torch.cat(padded_adv_fake_ids_gpt_list, dim=0)

if step % PRINT_INTERVAL == 0 : print(f"   {step} - G_ADV - adv_fake_ids_gpt_batch device: {adv_fake_ids_gpt_batch.device},

if adv_fake_ids_gpt_batch.size(1) <= prompt_len_adv:
    if step % PRINT_INTERVAL == 0: print(f"Warning too short")
else:

    adv_D_encodings = bert_tokenizer(adv_generated_texts,padding='max_length',truncation=True,max_length=BERT_MAX_LENGTH,ret
    adv_D_input_ids = adv_D_encodings.input_ids.to(DEVICE)
    adv_D_attention_mask = adv_D_encodings.attention_mask.to(DEVICE)

    with torch.no_grad():
        outputs_adv_d = discriminator(input_ids=adv_D_input_ids, attention_mask=adv_D_attention_mask)
        bert_logits_adv = outputs_adv_d.logits
        reward_adv = bert_logits_adv[:, 0].detach().to(DEVICE)
        reward_adv_val = reward_adv.mean().item()


    adv_seq_input_gpt = adv_fake_ids_gpt_batch[:, :-1].to(DEVICE)
    adv_seq_target_gpt = adv_fake_ids_gpt_batch[:, 1:].to(DEVICE)

    logits_adv_g, _ = model(adv_seq_input_gpt, targets=adv_seq_target_gpt)
    log_probs_adv_g = F.log_softmax(logits_adv_g, dim=-1)

    if step % PRINT_INTERVAL == 0 :
        print(f"   {step} - G_ADV - log_probs_adv_g device: {log_probs_adv_g.device}")
        print(f"   {step} - G_ADV - adv_seq_target_gpt device: {adv_seq_target_gpt.device}")

    gpt_vocab_size = model.config.vocab_size
    if adv_seq_target_gpt.max() >= gpt_vocab_size or adv_seq_target_gpt.min() < 0:
        if step % PRINT_INTERVAL == 0: print(f"(G_ADV):  {step}, adv_seq_target_gpt ")
    else:
        selected_log_probs = log_probs_adv_g.gather(2, adv_seq_target_gpt.unsqueeze(-1)).squeeze(-1)
        log_probs_of_generated_actions_adv = selected_log_probs[:, (prompt_len_adv - 1):] if prompt_len_adv > 0 else selecte

        if log_probs_of_generated_actions_adv.size(1) > 0:
            current_loss_adv = -(log_probs_of_generated_actions_adv.sum(dim=1) * reward_adv).mean()
            total_loss_g += BETA_ADV * current_loss_adv
            loss_adv_val = current_loss_adv.item()
        else:
            if step % PRINT_INTERVAL == 0: print(f"(G_ADV):步骤 {step}, log_probs_of_generated_actions_adv为空")
```

APPENDIX: BERT Classifier Structure (Embeddings)

```
BertForSequenceClassification(
            (bert): BertModel(
                        (embeddings): BertEmbeddings(
                                    (word_embeddings): Embedding(30522, 768, padding_idx=0)
                                    (position_embeddings): Embedding(512, 768)
                                    (token_type_embeddings): Embedding(2, 768)
                                    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                                    (dropout): Dropout(p=0.1, inplace=False)
                        )
```

APPENDIX: BERT Classifier Struture (Encodings)

```
(encoder): BertEncoder(
            (layer): ModuleList(
                        (0-11): 12 x BertLayer(
                        (attention): BertAttention(
                        (self): BertSdpaSelfAttention(
                                    (query): Linear(in_features=768, out_features=768, bias=True)
                                    (key): Linear(in_features=768, out_features=768, bias=True)
                                    (value): Linear(in_features=768, out_features=768, bias=True)
                                    (dropout): Dropout(p=0.1, inplace=False)
                        )
                        (output): BertSelfOutput(
                                    (dense): Linear(in_features=768, out_features=768, bias=True)
                                    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                                    (dropout): Dropout(p=0.1, inplace=False)
                        )
            )

            (intermediate): BertIntermediate(
                        (dense): Linear(in_features=768, out_features=3072, bias=True)
                        (intermediate_act_fn): GELUActivation() )
                        (output): BertOutput(
                                    (dense): Linear(in_features=3072, out_features=768, bias=True)
                                    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                                    (dropout): Dropout(p=0.1, inplace=False)
                        )
            )
))
```

APPENDIX: BERT Classifier Structure (Pooling & Classification Layer)

```
(pooler): BertPooler(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (activation): Tanh() )
)
(dropout): Dropout(p=0.1, inplace=False)
(classifier): Linear(in_features=768, out_features=2, bias=True) )
```